# Cost-Optimal Execution of Trees of Boolean Operators with Shared Streams

Henri Casanova, Lipyeow Lim, Yves Robert, Frédéric Vivien, Dounia Zaidouni


# Cost-Optimal Execution of Trees of Boolean Operators with Shared Streams

**Henri Casanova, Lipyeow Lim, Yves Robert, Frédéric Vivien, and Dounia Zaidouni**

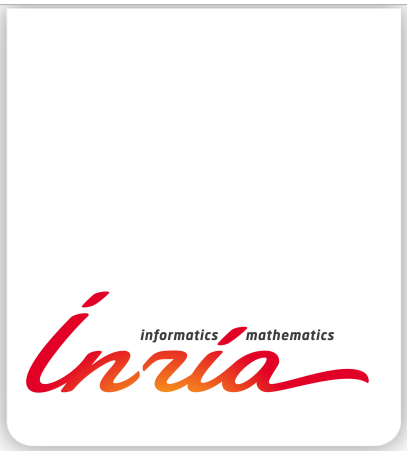
**RESEARCH REPORT**

**N° 8373**

October 2013

Project-Team Roma

ISSN 0249-6399 ISRN INRIA/RR--8373--FR+ENG




# Cost-Optimal Execution of Trees of Boolean Operators with Shared Streams

Henri Casanova*, Lipyeow Lim*, Yves Robert†‡, Frédéric Vivien§‡, and Dounia Zaidouni§‡

Project-Team Roma

Research Report n° 8373 — October 2013 — 36 pages

**Abstract:** The processing of queries expressed as trees of boolean operators applied to predicates on sensor data streams has several applications in mobile computing. Sensor data must be retrieved from the sensors to a query processing device, such as a smartphone, over one or more network interfaces. Retrieving a data item incurs a cost, e.g., an energy expense that depletes the smartphone's battery. Since the query tree contains boolean operators, part of the tree can be shortcircuited depending on the retrieved sensor data. An interesting problem is to determine the order in which predicates should be evaluated so as to minimize the expected query processing cost. This problem has been studied in previous work assuming that each data stream occurs in a single predicate. In this work we remove this assumption since it does not necessarily hold for real-world queries. Our main results are an optimal algorithm for single-level trees and a proof of NP-completeness for DNF trees. For DNF trees, however, we show that there is an optimal predicate evaluation order that corresponds to a depth-first traversal. This result provides inspiration for a class of heuristics. We show that one of these heuristics largely outperforms other sensible heuristics, including the one heuristic proposed in previous work for our general version of the query processing problem.

**Key-words:** query processing, boolean operators, energy, scheduling, greedy algorithm, data sharing

* University of Hawaii at Mānoa, HI, USA
† École normale supérieure de Lyon, France
‡ LIP laboratory – CNRS, ENS Lyon, INRIA, UCB Lyon 1
§ INRIA

**RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES**

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

# Exécution de coût optimal d'arbres d'opérateurs booléens partageant des données

**Résumé :** Le traitement de requêtes, exprimées sous forme d'arbres d'opérateurs booléens appliqués à des prédicats sur des flux de données de senseurs, a de nombreuses applications dans le domaine du calcul mobile. Les données doivent être transférées des senseurs vers l'appareil de traitement des données, par exemple un smartphone. Transférer une donnée induit un coût, par exemple une consommation énergétique qui diminuera la charge de la batterie du smartphone. Comme l'arbre de requêtes contient des opérateurs booléens, des pans de l'arbre peuvent être court-circuités en fonction des données récupérées. Un problème intéressant est de déterminer l'ordre dans lequel les prédicats doivent être évalués afin de minimiser l'espérance du coût du traitement de la requête. Ce problème a déjà été étudié sous l'hypothèse que chaque flux apparaît dans un seul prédicat. Dans le présent travail nous éliminons cette hypothèse qui ne correspond pas forcément à la réalité. Nos principaux résultats sont un algorithme optimal pour les arbres avec un seul niveau, et une preuve de NP-complétude pour les arbres sous forme normale disjonctive. Pour les arbres sous forme normale disjonctive, cependant, nous montrons qu'il existe un ordre optimal d'évaluation des prédicats qui correspond à un parcours en profondeur d'abord. Ce résultat nous sert à concevoir toute une classe d'heuristiques. Nous montrons que l'une de ces heuristiques a de bien meilleurs résultats que les autres heuristiques et, entre autres, que la seule heuristique précédemment proposée pour le cadre général.

**Mots-clés :** traitement de requêtes, opérateurs booléens, énergie, ordonnancement, algorithmique probabiliste, algorithme glouton, partage de données

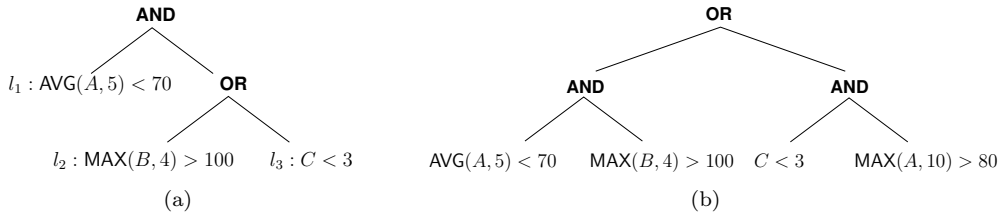



Figure 1: Two query tree examples: (a) a *read-once* query; (b) a *shared* query.

# 1 Introduction

There has been a recent explosion in the use of personal mobile devices for "mobile sensing" applications. For instance, smartphones are equipped with increasingly sophisticated sensors (e.g., GPS, accelerometer, gyroscope, microphone) that enable near real-time sensing of an individual's activity or environmental context. A smartphone can then perform embedded query processing on the sensor data streams, e.g., for social networking [5], remote health monitoring [6]. The continuous processing of streams, even when data rates are moderate (such as for GPS or accelerometer data), can cause commercial smartphone batteries to be depleted in a few hours [1]. It is thus crucial to reduce the amount of sensor data acquired for query processing, so as to reduce energy consumption and lengthen battery life.

In this work we study the problem of minimizing the expected sensor data acquisition cost (e.g., number of bytes, energy consumption due to byte transfers) when evaluating a query expressed as a tree of arbitrarily composed conjunctive and disjunctive boolean operators applied to boolean *predicates*. Each predicate is computed over data items from a particular data stream generated periodically by a sensor, and as a certain probability of evaluating to true. The evaluation of the query stops as soon as a truth value has been determined, possibly *shortcircuiting* part of the query tree. A "push" model by which sensors continuously transmit data to the device maximizes the amount of acquired data and is thus not practical. Instead, a "pull" model has been proposed [4], by which the query engine running on the device carefully chooses the *order* and the *numbers of data items* to request from each individual sensor. This choice is based on a-priori knowledge of operator costs and probabilities, which can be inferred based on historical traces obtained for previous query executions. In practice, such intelligent processing is possible thanks to the programming and data filtering capabilities that are emerging on many wearable sensor platforms (e.g., the SHIMMER platform [7]) so that data storage and transmission algorithms can be programmed "over the air."

Two example query trees are shown in Figure 1, assuming streams named $A$, $B$, and $C$, which are assumed to produce integer data items. Each leaf corresponds to a boolean predicate. A predicate may involve no operator, e.g., "$C < 3$" is true if the last item from stream $C$ is strictly lower than 3, or based on an arbitrary operator (in this example $\mathsf{MAX}$ or $\mathsf{AVG}$) which is applied to a time-window for a stream, e.g., "$\mathsf{AVG}(A, 5) < 70$" is true if the average of the last 5 items from $A$ is strictly lower than 70).

The problem of computing the truth value of a boolean query tree while incurring the minimum cost is known as Probabilistic AND-OR Tree Resolution (PAOTR) and has been studied extensively in the literature. In particular, [3] provides both a survey of known theoretical results and several new results, all assuming that each data stream occurs in at most one leaf of the query tree. This assumption is termed *read-once* therein. In this case, for AND-trees (i.e., single-level trees with an AND operator at the root node) a simple $O(n \log n)$ greedy algorithm

produces an optimal leaf evaluation order ($n$ is the number of leaves in the query tree) [8]. For DNF trees (i.e., collections of AND-trees whose roots are the children of a single OR node), a $O(n \log n)$ depth-first traversal of the trees that reuses the algorithm in [8] to order leaves within each AND produces an optimal evaluation order [3]. For general AND-OR-trees the complexity of the problem is open. The example query tree in Figure 1(a) is a *read-once* query since no stream occurs in two leaves.

By contrast, in this work we study the more general case, which we term *shared*, in which a stream can occur in multiple leaves. The example in Figure 1(b) corresponds to a *shared* case since stream $A$ occurs in two leaves. The device that processes the query acquires data items from streams and holds each data item in memory until that data item is no longer relevant. A data item from a stream is no longer relevant when it is older than the maximum time-window used for that stream in the query. Each time a leaf of the query must be evaluated, one can then compute the number of data items that must be retrieved from the relevant stream given the time-windows of the operator applied to that stream and the data items from that stream that are already in the device's memory. For example, considering the query in Figure 1(b), assume the predicate "$\mathsf{AVG}(A, 5) < 70$" is evaluated first, thus pulling 5 items from stream $A$. If later the predicate "$\mathsf{MAX}(A, 10) > 80$" needs to be evaluated then only 5 additional items must be pulled.

The *shared* scenario is important in practice, and has been introduced and investigated in [4]. In that work the authors do not give theoretical results, but instead develop heuristics to determine an order of operator evaluation that hopefully leads to low data acquisition costs. To the best of our knowledge, the complexity of the PAOTR problem in the *shared* case has never been addressed in the literature, likely because re-using stream data across leaves dramatically complicates the problem. When picking a leaf evaluation order, interdependences between the leaves must be taken into account. And in fact, even when a leaf evaluation order is given, computing the expected query cost is intricate while this same computation is trivial in the *read-once* case.

In this work we study the PAOTR problem in the *shared* case and make the following contributions:

- For AND-trees we give an optimal algorithm (which is much more involved than the optimal algorithm in the *read-once* case);
- For DNF trees we show that the problem is NP-complete; but we are able to prove that there exists an optimal leaf evaluation order that is depth-first;
- For DNF trees we develop heuristics that we evaluate in simulation and compare to the optimal solution (computed via an exhaustive search) and to the heuristic proposed in [4].

In Section 2 we discuss models, the problem statement, and related work. We study AND-trees and DNF trees in Section 3 and Section 4, respectively. Section 5 concludes the paper with a brief summary of our findings and perspectives on future work. Detailed proofs of some of our theoretical results are provided in appendices.

## 2 Problem Statement and Examples

To define our problem we reuse the formalism and terminology in [3]. A query is an AND-OR *tree*, i.e., a rooted tree whose non-leaf nodes are AND or OR operators, and whose leaf nodes are labeled with probabilistic boolean predicates. Each predicate is evaluated over data items generated by a data *stream*. The evaluation of each predicate has a known *success probability* (the probability that the predicate evaluates to TRUE) and a *cost*. In practice, the success probability can be estimated based on historical traces obtained from previous query evaluations. As in [3], we assume *independent* predicates, meaning that two predicates at two leaf nodes in a query

are statistically independent. The cost is determined by the number of data items required to perform the evaluation and the evaluation cost per data item for the stream. For instance, the cost of a data item could correspond to the energy cost, in joules, of acquiring one data item based on the communication medium used for the stream and the data item size.

More formally, we consider a set of $s$ streams, $\mathcal{S} = \{S_1, \ldots, S_s\}$. Stream $S_k$ has a cost per data item of $c(S_k)$. A query on these streams, $\mathcal{T}$, is a rooted AND-OR tree with $m$ leaves, $l_1, \ldots, l_m$. Leaf $l_j$ has success, resp. failure, probability $p_j$, resp. $q_j = 1 - p_j$, and requires the last $d_j$ items from stream $S(j) \in \mathcal{S}$. The objective is to compute the truth value of the root of the query tree by evaluating the leaves of the tree. Because each non-leaf node in a query tree is either an OR or an AND operator, it may not be necessary to evaluate all the leaves due to *shortcircuiting*. In other words, as soon as any child node of an OR, resp. AND, operator evaluates to TRUE, resp. FALSE, the truth value of the operator is known and can be propagated toward the root. For a given query, we define a *schedule* as an evaluation order of the leaves of the query tree, represented as a sorted sequence of the leaves.

We define the *cost* of a schedule as the **expected value** of the sum of the costs incurred for all leaves that are evaluated before the root's truth value is determined. For instance, consider the query in Figure 1(a), in which leaves are labeled $l_1, l_2, l_3$, and consider the schedule $l_2, l_3, l_1$. The query processing begins with the acquisition of the data items necessary for evaluating $l_2$, which has cost $4 \cdot c(B)$. With probability $p_2$, $l_2$ evaluates to TRUE, thus shortcircuiting the evaluation of $l_3$. Therefore, the expected evaluation cost of the OR operator is: $4 \cdot c(B) + q_2 \cdot c(C)$. If the OR operator evaluates to FALSE, which happens with probability $q_2 q_3$, then the evaluation of $l_1$ is shortcircuited. Otherwise, $l_1$ must be evaluated. The overall cost of the schedule is thus: $4 \cdot c(B) + q_2 \cdot c(C) + (1 - q_2 q_3) \cdot 5 \cdot c(A)$. Recall that this query tree is for a *read-once* scenario.

The PAOTR problem consists in determing a schedule with minimum cost. The complexity of this problem is unknown in the *read-once* case for general AND-OR trees, while optimal polynomial-time algorithms are known for AND-trees [8] and DNF trees [3]. In this work, we focus on these two types of trees in the *shared* case, seeking to develop optimal algorithms or to show NP-completeness. We refer the reader to [3] for a detailed review of the PAOTR literature. To the best of our knowledge, the only work that has studied the *shared* case is [4], in which a heuristic is proposed for DNF trees. We evaluate this heuristic in Section 4.4. In the next two sections we give examples of cost computations for an AND-tree and a DNF tree both in the *shared* case.

## 2.1 AND-tree example

Consider the AND-tree query depicted in Figure 2 with three leaves labeled $l_1$, $l_2$, and $l_3$, for two streams $A$ and $B$. For each leaf ($l_i$), we indicate the stream ($S(i)$), the number of data items needed from that stream to evaluate the leaf ($d_i$), and the success probability ($p_i$). For instance, leaf $l_2$ requires $d_2 = 2$ items from stream $S(2) = A$ and evaluates to TRUE with probability $p_2 = 0.1$. We assume that retrieving a data item from a stream has unitary cost, regardless of

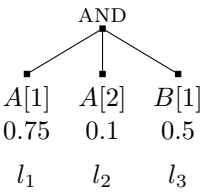



Figure 2: Example *shared* AND-tree.

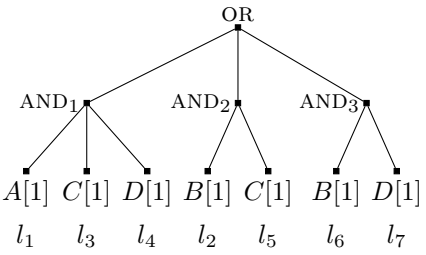



Figure 3: Example DNF tree.

the stream. There are 6 possible schedules for this tree, each schedule corresponding to one of the 3! orderings of the leaves. The optimal algorithm for *read-once* AND-trees sorts the leaves by non-decreasing $d_j c(S(j))/q_j$ [8]. Because $\frac{1\times c(A)}{q_1} = \frac{1}{1-0.75} = 4$, $\frac{2\times c(A)}{q_2} = \frac{2}{1-0.1} \approx 2.22$, and $\frac{1\times c(B)}{q_3} = \frac{1}{1-0.5} = 2$, this algorithm schedules leaf $l_3$ first. There are two possible schedules with $l_3$ as the first leaf:

- $l_3$, $l_1$, $l_2$ whose cost is: $c(B) + p_3 \times (c(A) + p_1 \times c(A)) = 1 + 0.5 \times (1 + 0.75 \times 1) = 1.875$; and
- $l_3$, $l_2$, $l_1$ whose cost is: $c(B) + p_3 \times (2 \times c(A) + p_2 \times 0 \times c(A)) = 1 + 0.5 \times (2 + 0.1 \times 0) = 2$.

However, another schedule, $l_1$, $l_2$, $l_3$, has a lower cost: $c(A) + p_1 \times (c(A) + p_2 \times c(B)) = 1 + 0.75 \times (1 + 0.1 \times 1) = 1.825$. Therefore, the optimal algorithm for the PAOTR problem for *read-once* AND-trees is no longer optimal in the *shared* case.

## 2.2 DNF tree example

Figure 3 shows a DNF tree with three AND nodes, for four streams $A$, $B$, $C$, and $D$. Each leaf requires only one data item from a stream. Leaves are labeled $l_1$ to $l_7$, in the order in which they appear in a given schedule. This example is meant to illustrate the difficulty of the PAOTR problem in the case of DNF trees in the *shared* scenario. In particular, computing the cost of a schedule is much more complicated than in the *read-once* scenario due to inter-leaf dependencies. Let $\mathcal{C}_j$ be the cost of evaluating leaf $l_j$, and $\mathcal{C}$ the overall cost of the schedule. We consider the 7 leaves one by one, in order:

**Leaf** $l_1$ – The first leaf is evaluated: $\mathcal{C}_1 = c(A)$.

**Leaf** $l_2$ – This is the first leaf in its AND, no AND has been fully evaluated so far, and $l_2$ is the first encountered leaf that requires stream $B$. Therefore, $l_2$ is always evaluated, requiring a data item from stream $B$: $\mathcal{C}_2 = c(B)$.

**Leaf** $l_3$ – This is the second leaf from its AND, no AND has been fully evaluated so far, and $l_3$ is the first encountered leaf that requires stream $C$. Therefore, a data item from $C$ is acquired if and only if $l_1$ evaluates to TRUE: $\mathcal{C}_3 = p_1 c(C)$.

**Leaf** $l_4$ – This is the third leaf from its AND, no AND has been fully evaluated so far, and $l_4$ is the first encountered leaf that requires stream $D$. Therefore, one data item is acquired from $D$ if and only if $l_1$ and $l_3$ both evaluate to TRUE: $\mathcal{C}_4 = p_1 p_3 c(D)$.

**Leaf** $l_5$ – This is the second leaf from its AND, and $\text{AND}_1$ has been fully evaluated so far. However, one of the leaves of that AND, $l_3$, requires a data item that is also needed by $l_5$, from stream $C$. If $l_3$ has been evaluated, then the evaluation cost of $l_5$ is 0 because the necessary data item from $C$ has already been acquired and is available "for free" when evaluating $l_5$. If $l_3$ has not been evaluated (with probability $1 - p_1$), it means that $\text{AND}_1$ has evaluated to FALSE. Then, if $l_2$ has evaluated to TRUE, $l_5$ must be evaluated thus requiring the data item from

stream $C$. We obtain $\mathcal{C}_5 = (1 - p_1)p_2c(C)$.
**Leaf** $l_6$ – Since $l_2$ is always evaluated the data item from stream $B$ required by $l_6$ is always available for free: $\mathrm{C}_6 = 0$.
**Leaf** $l_7$ – This is the second leaf from its AND, and $\mathrm{AND}_1$ and $\mathrm{AND}_2$ have been fully evaluated so far. However, one of the leaves of $\mathrm{AND}_1$, $l_4$, but none of those of $\mathrm{AND}_2$, require the data item that is needed by $l_7$ from stream $D$. Therefore, $l_7$ must be evaluated and its evaluation is not free if and only if $l_4$ has not been evaluated, $\mathrm{AND}_2$ has evaluated to FALSE, and the evaluation of $\mathrm{AND}_3$ went as far as $l_7$. Therefore, $\mathcal{C}_7 = (1 - p_1p_3)(1 - p_2p_5)p_6c(D)$.

Overall, we obtain the cost of the schedule:

$$\begin{aligned}\mathcal{TC} &= c(A) + c(B) + (p_1 + (1 - p_1)p_2)c(C) \\ &+ (p_1p_3 + (1 - p_1p_3)(1 - p_2p_5)p_6)c(D)\end{aligned}$$

Given the complexity of the above cost computation, one might expect the PAOTR problem to be NP-complete in the *shared* case (recall that it is polynomial in the *read-once* case). We confirm this expectation in Section 4.

# 3 AND trees

In this section, we focus on AND-trees. We have seen in Section 2.1 that the simple greedy algorithm proposed in [8] in the *read-once* case is not optimal in the *shared* case. We propose an algorithm and we prove that it is optimal. This algorithm is still greedy but compares the ratios of cost to failure probability of all sequences of leaves that use the same stream, instead of only considering pair-wise leaf comparisons. We begin in Section 3.1 with a preliminary result on the optimal ordering of leaves that use the same stream.

## 3.1 Ordering same-stream leaves

In the example given in Section 2.1, we considered two schedules that begin with leaf $l_3$. In the first schedule leaf $l_1$ precedes $l_2$, while the converse is true in the second schedule. Leaf $l_1$ requires one data item from stream $A$, while leaf $l_2$ requires two data items from the same stream. Therefore the first schedule is always preferable to the second schedule: if we evaluate $l_1$ before $l_2$ and if $l_1$ evaluates to FALSE, then there is no need to retrieve the second data item and the cost is lowered. A general result can be obtained:

**Proposition 1.** *Consider an* AND*-tree and a leaf $l_i$ that requires $d_i$ data items from a stream $S$. In an optimal schedule $l_i$ is scheduled before any leaf $l_j$ that requires $d_j > d_i$ data items from stream $S$.*

*Proof.* See Appendix A for the proof. □

## 3.2 Optimal schedule

Consider an AND-tree with $m$ leaves, $l_1, \ldots, l_m$, for $s$ streams, $S_1, \ldots, S_s$. We define $\mathcal{L}_k = \{l_j | S(l_j) = S_k\}$, i.e., the set of leaves that require data items from stream $S_k$. Algorithm 1 shows a greedy algorithm (implemented recursively for clarity of presentation) that takes as input the $\mathcal{L}_k$ sets, an initially empty schedule $\xi$, and an array of $s$ integers, *NItems*, whose elements are all initially set to zero. This array is used to keep track, for each stream, of how many data items from that stream have been retrieved in the schedule so far. Each call to the algorithm

appends to the schedule a sequence of leaves that require data items from the same stream, in increasing order of number of data items required. The algorithm stops when all leaves have been scheduled. The algorithm first loops through all the streams (the $k$ loop). For each stream, the algorithm then loops over all the leaves that use that stream, taken in increasing order of the number of items required. For each such leaf the algorithm computes the ratio (variable *Ratio*) of cost to probability of failure of the sequence of leaves up to that leaf. The leaf with minimum such ratio is selected (leaf $l_{j_0}$ in the algorithm, which requires $d_{j_0}$ data items from stream $S(l_{j_0})$). In the last loop of the algorithm, all unscheduled leaves that require $d_{j_0}$ or fewer data items from stream $S(l_{j_0})$ are appended to the schedule in increasing order of the number of required data items.

**Algorithm 1:** GREEDY$(\{\mathcal{L}_1, ..., \mathcal{L}_s\}, \xi, NItems)$

**if** $\cup_{i=1}^{s} \mathcal{L}_i = \varnothing$ **then return** $\xi$ $MinRatio \leftarrow +\infty$
**for** $k = 1$ *to* $s$ **do** loop on streams
  $Cost \leftarrow 0$
  $Proba \leftarrow 1$
  $Num \leftarrow NItems[k]$
  **for** $l_j$ *in* $\mathcal{L}_k$ *by increasing* $d_j$ **do**
    $Cost \leftarrow Cost + Proba \times (d_j - Num) \times c(k)$
    $Proba \leftarrow Proba \times p_j$
    $Num \leftarrow d_j$
    $Ratio \leftarrow \frac{Cost}{(1-Proba)}$
    **if** $Ratio < MinRatio$ **then**
      $MinRatio \leftarrow Ratio$
      $j_0 \leftarrow j$
**for** $l_j$ *in* $\mathcal{L}_{S(j_0)}$ *by increasing* $d_j$ **do**
  **if** $d_j \leq d_{j_0}$ **then**
    $\xi$.append$(l_j)$
    $\mathcal{L}_{S(j_0)} \leftarrow \mathcal{L}_{S(j_0)} \setminus \{l_j\}$
$NItems[S(j_0)] \leftarrow d_{j_0}$
**return** GREEDY $(\{\mathcal{L}_1, ..., \mathcal{L}_s\}, \xi, NItems)$

**Theorem 1.** *Algorithm 1 is optimal for the* shared *PAOTR problem for* AND*-trees.*

*Proof.* See Appendix B for the proof. □

One may wonder how the optimal algorithm in the *read-once* case [8], which simply sorts the leaves by increasing $d_j c(S(j))/q_j$, fares in the *shared* case. In other terms, is Algorithm 1 really needed in practice? Figure 4 shows results for a set of randomly generated AND-trees. We define the *sharing ratio*, $\rho$, of a tree as the expected number of leaves that use the same stream, i.e., the total number of leaves divided by the number of streams. For a given number of leaves $m = 2, \ldots, 20$ and a given sharing ratio $\rho = 1, 5/4, 4/3, 3/2, 2, 3, 4, 5, 10$, we generate 1,000 random trees for a total of 157,000 random trees (note that $\rho$ cannot be larger than the number of leaves). Leaf success probabilities, numbers of data items needed at each leaf, and per data item costs are sampled from uniform distributions over the intervals $[0, 1]$, $[1, 5]$, and $[1, 10]$, respectively. For each tree we compute the cost achieved by the algorithm in [8] and

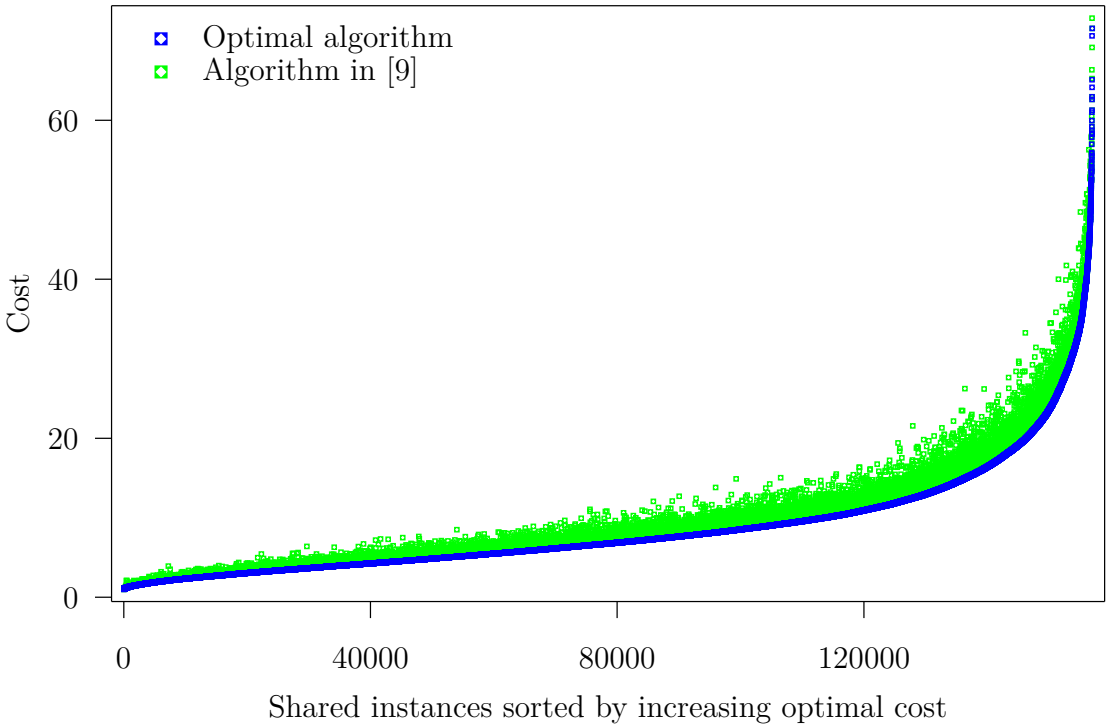



Figure 4: Cost achieved by the algorithm in [8] and that achieved by the optimal algorithm, shown for each of the 157,000 AND-tree instances sorted by increasing optimal cost.

that achieved by our optimal algorithm. Figure 4 plots these costs for all instances, sorted by increasing optimal cost. Due to this sorting, the large number of samples, and the limited resolution, the set of points for the optimal algorithm appears as a curve while the set of points for the algorithm in [8] appears as a cloud of points. The algorithm in [8] can lead to costs up to 1.86 times larger than the optimal. It leads to costs more than 10% larger for 19.54% of the instances, and more than 1% larger for 60.20% of the instances. The two algorithms lead to the same cost for 11.29% of the instances. We conclude that, in the *shared* case, Algorithm 1 provides substantial improvements over the optimal algorithm for the *read-once* case.

# 4 DNF Trees

In this section we consider DNF trees. First, in Section 4.1 we provide a method for computing the expected cost of a given schedule for a DNF tree. In Section 4.2 we show that depth-first schedules are dominant, which means that there always exists a depth-first schedule that is optimal. In Section 4.3, we then prove that the problem is NP-complete. This is in sharp contrast with the *read-once* case, in which a simple greedy algorithm is optimal [3]. In Section 4.4 we propose several heuristics to schedule a DNF tree and evaluate their performance on randomly generated problem instances.

### 4.1 Evaluation of a schedule

We have seen in Section 2.2 in an example that computing the cost of a schedule is non-trivial for DNF trees. In this section we formalize this computation. Consider a DNF tree with $N$ AND nodes, indexed $i = 1, \ldots, N$. AND node $i$ has $m_i$ leaves, denoted by $l_{i,j}$, $j = 1, \ldots, m_i$. The probability of success of leaf $l_{i,j}$ is denoted by $p_{i,j}$, and the stream that leaf $l_{i,j}$ requires is denoted by $S(i,j)$. We use $\mathcal{L}$ to denote the set of all the leaves. We consider a schedule $\xi$, which is an ordering of the leaves, and use $l_{s,t} \prec l_{u,v}$ to indicate that leaf $l_{s,t}$ occurs before leaf $l_{u,v}$ in $\xi$. We consider that the query is over $s$ streams, $S_k$, $k = 1, \ldots, s$. The cost per data item of $S_k$ is denoted by $c(S_k)$. We define the "$t$-th data item" of a stream as the data item produced $t$ time-steps ago, so that the first data item is the one produced most recently, the second is the one produced before the first, etc. In this manner, when we say that a leaf $l_{i,j}$ requires $d_{i,j}$ data items it means that it requires all $t$-th data items of the stream for $t = 1, 2, \ldots, d_{i,j}$.

Given the above, we define $\mathcal{L}_{k,t}$ as the set of the leaves that require the $t$-th data item from stream $S_k$, and that are the first of their respective AND nodes to require that data item. Formally, we have:

$$\mathcal{L}_{k,t} = \left\{ l_{i,j} \in \mathcal{L} \;\middle|\; \begin{array}{c} S(i,j) = S_k,\ d_{i,j} \geq t, \text{ and} \\ \forall r \neq j, S(i,r) \neq S_k \text{ or } d_{i,r} < t \\ \text{or } l_{i,j} \prec l_{i,r} \end{array} \right\}$$

We also define $\mathcal{A}_{i,j}$, the index set of all AND nodes that have been fully evaluated before a leaf $l_{i,j}$ is evaluated, as:

$$\mathcal{A}_{i,j} = \{k \mid m_k = |\{l_{k,r} | l_{k,r} \prec l_{i,j}\}|\}.$$

If we use $\mathcal{C}_{i,j,t}$ to denote the expected cost of retrieving the $t$-th data item of the relevant stream when evaluating leaf $l_{i,j}$, then the total cost $\mathcal{C}$ of the schedule $\xi$ is:

$$\mathcal{C} = \sum_{i=1}^{N} \sum_{j=1}^{m_i} \sum_{t=1}^{d_{i,j}} \mathcal{C}_{i,j,t}.$$

The following proposition gives $\mathcal{C}_{i,j,t}$.

**Proposition 2.** *Given a leaf $l_{i,j}$ that requires the $t$-th data item from stream $S_k$, if there exists $r$ such that $l_{i,r} \prec l_{i,j}$ and $l_{i,r} \in \mathcal{L}_{k,t}$, then $\mathcal{C}_{i,j,t} = 0$. Otherwise:*

$$\begin{aligned} \mathcal{C}_{i,j,t} = & \prod_{\substack{l_{r,s} \in \mathcal{L}_{k,t} \\ l_{r,s} \prec l_{i,j}}} \left(1 - \prod_{l_{r,u} \prec l_{r,s}} p_{r,u}\right) \\ & \times \prod_{\substack{a \in \mathcal{A}_{i,j} \\ \nexists r,\ l_{a,r} \in \mathcal{L}_{k,t}}} \left(1 - \prod_{r=1}^{m_a} p_{a,r}\right) \\ & \times \left(\prod_{l_{i,u} \prec l_{i,j}} p_{i,u}\right) \times c(S(i,j)). \end{aligned}$$

*Proof.* See Appendix C for the proof. □

## 4.2 Dominance of depth-first schedules

**Theorem 2.** *Given a* DNF *tree, there exists an optimal schedule that is depth-first, i.e., that processes* AND *nodes one by one.*

*Proof.* Consider a DNF tree $\mathcal{T}$ and a schedule $\xi$. Without loss of generality we assume that the AND nodes, $A_1, \ldots, A_n$, are numbered in the order of their completion. Thus, according to $\xi$, $A_1$ is the first AND node with all its leaves evaluated. We denote by $M$ the number (possibly zero) of AND nodes that $\xi$ processes one by one and entirely at the start of its execution. Therefore, if $\xi$ evaluates a leaf $l_{i,j}$, with $i \neq 1$, in the $m_1$ first steps, then $M = 0$. Finally, we assume that the leaves of an AND node are numbered according to their evaluation order in $\xi$.

We prove the theorem by contradiction. Let us assume that there does not exist a schedule that satisfies the desired property. Let $\xi$ be an optimal schedule that maximizes $M$. By definition of $M$ and by the hypothesis on the numbering of the AND nodes, schedule $\xi$ evaluates some leaves of the AND nodes $A_{M+2}$, ..., $A_n$ before it evaluates the last leaf of $A_{M+1}$. Let $\mathcal{L}$ denote the set of these leaves. We now define a new $\xi'$ which starts by executing at least $M+1$ AND nodes one by one:

- $\xi'$ starts by evaluating the first $M$ AND nodes one by one, evaluating their leaves in the same order and at the same steps as in $\xi$;
- $\xi'$ then evaluates all the leaves of $A_{M+1}$ in the same order as in $\xi$ (but not at the same steps);
- $\xi'$ then evaluates the leaves in $\mathcal{L}$ in the same order as in $\xi$ (but not at the same steps);
- $\xi'$ finally evaluates the remaining leaves in the same order and at the same steps as in $\xi$.

The cost of a schedule is the sum, over all potentially acquired data items, of the cost of acquiring each data item times the probability of acquiring it. Let $d$ be a data item potentially needed by a leaf in $\mathcal{T}$. We show that the probability of acquiring $d$ is not greater with $\xi'$ than with $\xi$. We have three cases to consider.

**Case 1)** $d$ is not needed by a leaf of $A_{M+1}$ and not needed by a leaf in $\mathcal{L}$. Then $d$'s probability to be acquired is the same with $\xi$ and $\xi'$.

**Case 2)** $d$ is needed by at least one leaf of $A_{M+1}$. The only way in which a leaf that is evaluated in $\xi$ would not be evaluated in $\xi'$ is if $A_{M+1}$ evaluates to TRUE. By assumption, however, at least one leaf of $A_{M+1}$ uses $d$. Therefore, for $A_{M+1}$ to evaluate to TRUE, $d$ must be acquired. Consequently, the probability that $d$ is acquired is the same with $\xi$ and with $\xi'$.

**Case 3)** $d$ is needed by at least one leaf in $\mathcal{L}$ but not needed by any leaf of $A_{M+1}$. $\xi$ and $\xi'$ define the same ordering on the leaves in $\mathcal{L}$. For each AND node $A_i$, with $M+2 \leq i \leq N$, there is at most one leaf in $A_i \cap \mathcal{L}$ that can be the leaf responsible for the acquisition of $d$ with $\xi$, and it is the same leaf with $\xi'$. Let $\mathcal{F}$ be the set of all these leaves. Then, with $\xi$, the leaves in $\mathcal{F}$ are responsible for the acquisition of $d$ if and only if:

- $A_1$, ..., $A_M$ all evaluate to FALSE;
- None of the evaluated leaves of $A_1$, ..., $A_M$ needs $d$; and
- At least one of the leaves in $\mathcal{F}$ is evaluated.

Let us denote by $\mathcal{P}$ the probability that all the AND nodes $A_1$, ..., $A_M$ evaluate to FALSE and that none of the evaluated leaves of these AND nodes needs the data item $d$. Let us denote by $\mathcal{D}$ the probability that $d$ is acquired because of the evaluation of one of the leaves of the AND nodes $A_1$, ..., $A_M$. Finally, let $\mathcal{R}$ be the probability that one of the leaves evaluated with $\xi$ after $l_{M+1,m_{M+1}}$ acquires $d$, knowing that no leaves of $A_1$, ..., $A_M$ or in $\mathcal{L}$ acquires it. Then, with $\xi$, the probability $p$ that $d$ is acquired is:

$$p = \mathcal{D} + \mathcal{P}\left(1 - \prod_{l_{i,j} \in \mathcal{F}} \left(1 - \prod_{k=1}^{j-1} p_{i,k}\right)\right) + \mathcal{R} \tag{1}$$

because leaf $l_{i,j}$ is evaluated with probability $\prod_{k=1}^{j-1} p_{i,k}$, that is, if all the leaves from the same AND node that are evaluated prior to it all evaluate to TRUE. The second term of Equation (1) is the probability that the leaves in $\mathcal{F}$ are responsible for acquiring $d$.

With schedule $\xi'$, the leaves of $\mathcal{F}$ are responsible for the acquisition of $d$ if and only if:

- The AND nodes $A_1$, ..., $A_M$, and $A_{M+1}$ all evaluate to FALSE;
- None of the evaluated leaves of the AND nodes $A_1$, ..., $A_M$ need $d$; and
- At least one of the leaves in $\mathcal{F}$ is evaluated.

Thus, with $\xi'$, the probability $p'$ that $d$ is acquired is:

$$p' \quad = \quad \mathcal{D} \quad + \quad \mathcal{P}\left(1 - \prod_{k=1}^{m_{M+1}} p_{M+1,k}\right) \quad \times \quad \left(1 - \prod_{l_{i,j} \in \mathcal{F}} \left(1 - \prod_{k=1}^{j-1} p_{i,k}\right)\right) \quad + \quad \mathcal{R}$$

Comparing this equation with Equation 1, we see that $p'$ is not greater than $p$.

The probability that a data item is acquired with $\xi'$ is thus not greater than with $\xi$. Therefore, in each of the three cases the cost of $\xi'$ is not greater than the cost of $\xi$, meaning that $\xi'$ is also an optimal schedule. Since $\xi'$ starts by executing at least $M+1$ AND nodes one by one, we obtain a contradiction with the maximality assumption on $M$, which concludes the proof. □

## 4.3 NP-completeness

In the *read-once* case, an optimal algorithm for DNF trees is built on top of the optimal algorithm for AND-trees [3]. The same approach cannot be used in the *shared* case, as seen in a simple counter-example (see Appendix D). And, in fact, in this section we show the NP-completeness of finding an optimal schedule to evaluate a DNF tree.

**Definition 1** (DNF-Decision)**.** *Given a* DNF *tree and a cost bound $K$, is there a schedule whose expected cost does not exceed $K$?*

**Theorem 3.** DNF-Decision *is NP-complete.*

*Proof.* The NP-completeness is obtained via a non-trivial reduction from 2-PARTITION [2]. See the full proof in Appendix E. □

## 4.4 Heuristics

Given the NP-completeness result in the previous section, we now propose several polynomial-time heuristics for computing a schedule. These heuristics fall into three categories, which we term leaf-ordered, AND-ordered, and stream-ordered.

**Leaf-ordered heuristics** simply sort the leaves according to leaf costs ($\mathcal{C}$), failure probabilities ($q = 1 - p$), or the ratio of the two, which leads to three heuristics plus a baseline random one:

- Leaf-ordered, decreasing $q$ (prioritizes leaves with high chances of shortcutting the evaluation of an AND node);
- Leaf-ordered, increasing $\mathcal{C}$ (prioritizes leaves with low costs);
- Leaf-ordered, increasing $\mathcal{C}/q$ (prioritizes leaves with low costs and also with high chances of shortcutting the evaluation of an AND node);
- Leaf-ordered, random (baseline).

The above first three heuristics have intuitive rationales. Other options are possible (e.g., sort leaves by decreasing $\mathcal{C}$) but are easily shown to produce poor results in practice.

**AND-ordered heuristics**, unlike leaf-ordered heuristics, account for the structure of the DNF tree by building depth-first schedules, with the rationale that there is a depth-first schedule that is

optimal (Theorem 2). Furthermore, Algorithm 1 provides a way to compute an optimal schedule for the leaves within the same AND node. For this optimal schedule one can compute the (expected) cost and the probability of success of the AND node using the method in Section 4.1 (this method actually applies to more general DNF trees). Therefore, AND-ordered heuristics simply order the AND nodes based on their computed costs ($\mathcal{C}$), computed probability of success ($p$), or ratio of the two, and using Algorithm 1 for scheduling the leaves of each AND node, leading to three heuristics:

- AND-ordered, decreasing $p$ (prioritizes AND's with high chances of shortcircuiting the evaluation of the OR node);
- AND-ordered, increasing $\mathcal{C}$ (prioritizes AND's with low costs);
- AND-ordered, increasing $\mathcal{C}/p$ (prioritizes AND's with low costs and also with high chances of shortcircuiting the evaluation of the OR node);

There are two approaches to compute the cost of an AND node: (i) consider the AND node in isolation assuming that the OR node has a single AND node child; or (ii) account for previously scheduled AND nodes whose evaluation has caused some data items to be acquired with some probabilities. We terms the first approach "static" and the second approach "dynamic," giving us two versions of the last two heuristics above.

**Stream-ordered heuristics** proceed by ordering the streams from which data items are acquired, acquiring all items from a stream before proceeding to the next stream, until the truth value of the OR node has been determined. This idea was proposed in [4], and to the best of our knowledge it is the only previously proposed heuristic for solving the PAOTR problem in the *shared* scenario for DNF trees. For each stream $S$ the heuristic computes a metric, $R(S)$, defined as follows:

$$R(S) = \frac{\sum_{i,j|S(i,j)=S} q_{i,j} n_{i,j}}{\max_{i,j|S(i,j)=S} d_{i,j} c(S)},$$

where $n_{i,j}$ is the number of leaf nodes whose evaluation would be shortcircuited if leaf $l_{i,j}$ was to evaluate to FALSE. The numerator can thus be interpreted as the shortcutting power of stream $S$. The denominator is the maximum data element acquisition cost over all the leaves that use stream $S$. The heuristic orders the streams by increasing $R$ values. The rationale is that one should prioritize streams that can shortcut many leaf evaluations and that have low maximum data item acquisition costs. The heuristic as it is described in [4] acquires the maximum number of needed data items from each stream so as to compute truth values of all the leaves that require data items from that stream. In other words, the leaves that require data items from the same stream are scheduled in decreasing $d_{i,j}$ order. However, Proposition 1 holds for DNF trees, showing that it is always better to schedule these leaves in *increasing* $d_{i,j}$ order. We use this leaf order to implement this heuristic in this work. We have verified in our experiments that this version outperforms the version in [4] in the vast majority of the cases, with all remaining cases being ties.

In total, we consider 4 leaf-ordered, 5 AND-ordered, and 1 stream-ordered heuristics. We first evaluate these heuristics on a set of "small" instances for which we can compute optimal schedules using an exponential-time algorithm that performs an exhaustive search. Such an algorithm is feasible because, due to Theorem 2, it only needs to search over all possible depth-first schedules. Small instances are generated using the same method as that described in Section 3.2 for generating AND-tree instances. We generate DNF trees with $N = 2, \ldots, 9$ AND nodes and up to at most 20 leaves in total, generating 100 random instances for each configuration, for a total of 21,600 instances (The source code is available at www.ens-lyon.fr/LIP/ROMA/Data/DataForRR-8373.tgz). For each instance we compute the ratio between the cost achieved by each heuristic and the optimal cost. Figure 5 shows for each heuristic the ratio vs. the fraction of the instances for which the heuristic achieves a lower ratio. For

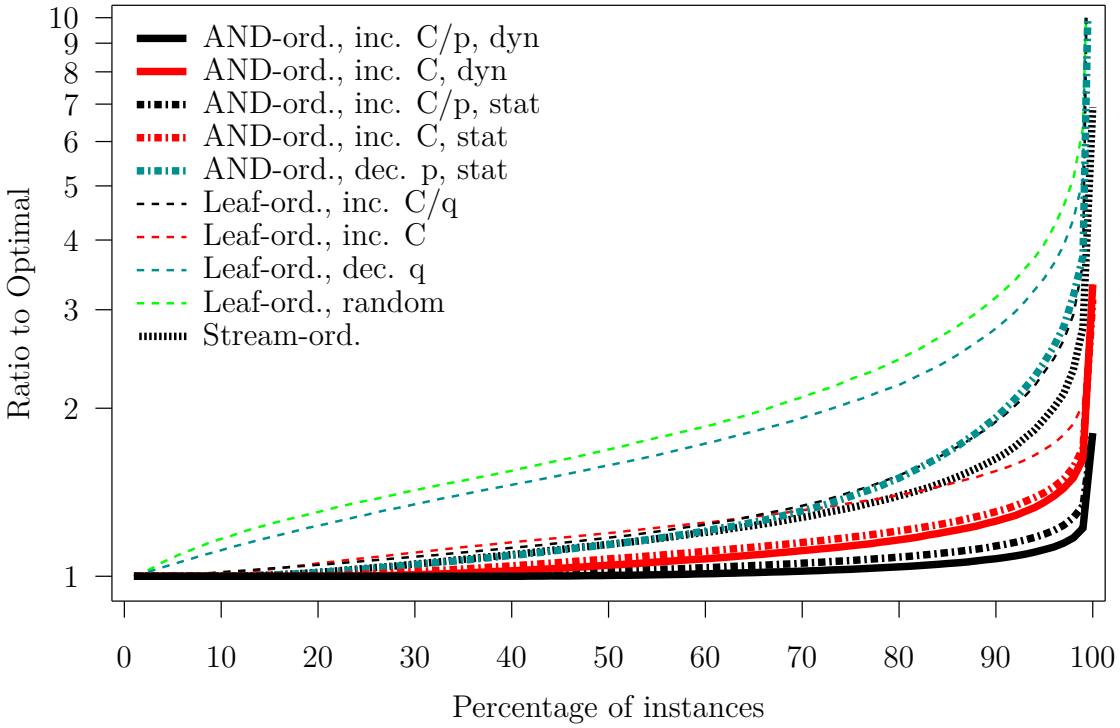



Figure 5: Ratio to optimal vs. fraction of the instances for which a smaller ratio is achieved, computed over the 21,600 "small" DNF tree instances.

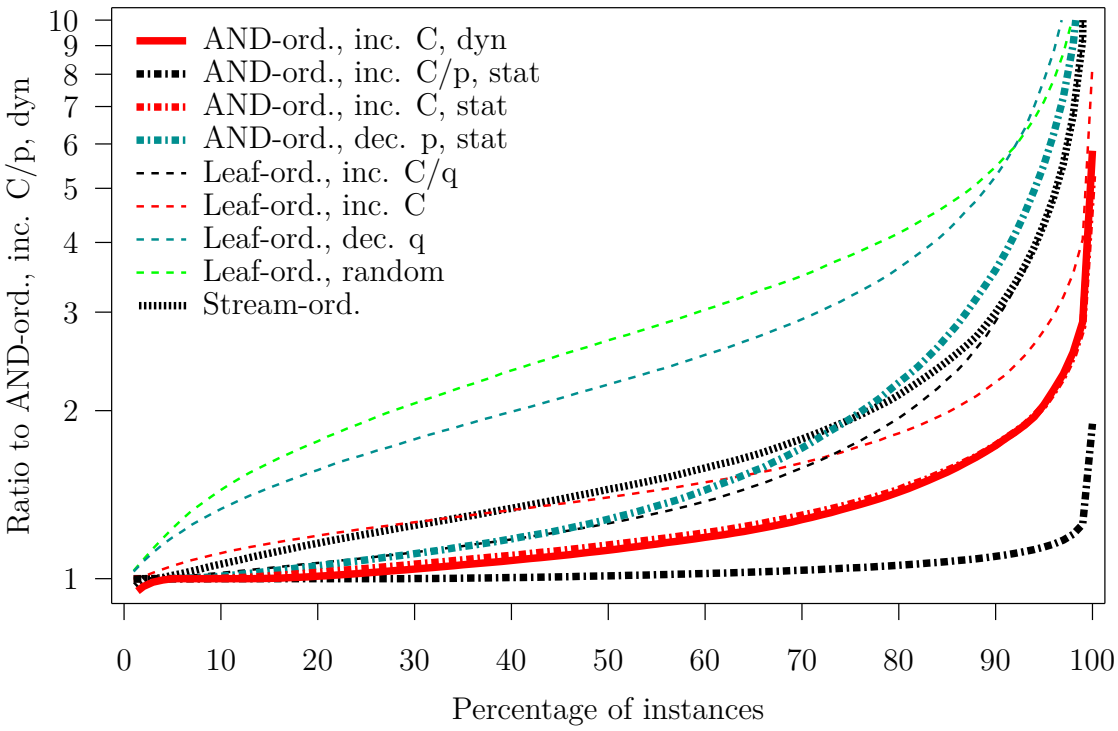



Figure 6: Ratio to AND-ordered increasing $\mathcal{C}/p$ dynamic vs. fraction of the instances for which a smaller ratio is achieved, computed over the 32,400 "large" DNF tree instances.

instance, a point at $(80, 2)$ means that the heuristic leads to schedules that are within a factor 2 of optimal for 80% of the instances, and more than a factor 2 away from optimal for 20% of the instances. The better the heuristic the closer its curve remains to the horizontal axis.

The trends in Figure 5 are clear. Overall the poorest results are achieved by the leaf-ordered heuristics, with the random such heuristic expectedly being the worst and the increasing $\mathcal{C}$ the best. The AND-ordered heuristics, save for the decreasing $p$ version, lead to the best results overall. More precisely, the best results are achieved by sorting AND's by increasing $\mathcal{C}/p$, with sorting by increasing $\mathcal{C}$ leading to the second-best results. For the two AND-ordered heuristics that have both a static and a dynamic version, the dynamic version leads to marginally better results than the static version. Finally, the stream-ordered heuristic leads to poorer results than the best leaf-ordered heuristics, and thus significantly worse than the best AND-ordered heuristics.

We also evaluate the heuristics on a set of "large" instances with $N = 2, \ldots, 10$ AND nodes and $m = 5, 10, 15, 20$ leaves per AND node, with 100 random instances per configuration, for a total of 32,400 instances. For most of these instances we cannot tractably compute the optimal cost. Consequently, we compute ratios to the cost achieved by the AND-ordered by increasing $\mathcal{C}/p$ dynamic heuristic, which leads to the best results for small instances. Results are shown in Figure 6. Essentially, all the observations made on the results for small instances still hold. We conclude that the best approach is to build a depth-first schedule, to sort the AND nodes by the ratio of their costs to probability of success, and to compute these costs dynamically, accounting for previously scheduled AND nodes.

# 5 Conclusion

Motivated by a query processing scenario for sensor data streams, we have studied a version of the Probabilistic And-Or Tree Resolution (PAOTR) problem [3] in which a data stream can be referenced by multiple leaves. We have given an optimal algorithm in the case of AND-trees and have shown NP-completeness in the case of DNF trees. For DNF we have shown that there is an optimal solution that corresponds to a depth-first traversal of the tree. This observation provides inspiration for a heuristic that largely outperforms the heuristic previously proposed in [4].

A possible future direction is to consider so-called *non-linear strategies* [3]. Although in this work we have considered a schedule as a leaf ordering (called a *linear strategy* in [3]), a more general notion is that of a decision tree in which the next leaf to be evaluated is chosen based on the truth value of the previous evaluated leaf. A practical drawback of a non-linear strategies is that the size of the description is exponential in the number of tree leaves. In [3], it is shown that in the *read-once* case linear strategies are dominant for DNF trees, meaning that there is always one optimal strategy that is linear. Via a simple counter example it can be shown that this is no longer true in the *shared* case (see Appendix F), thus motivating the investigation of non-linear strategies. Another possible future direction is to consider a less restricted version of the problem in which a single predicate at a leaf can access multiple streams rather than just a single one (e.g., "$AVG(X < 10) \geq MIN(Y, 20)$"). There is no reason for real-world queries to be limited to a single stream per predicate. An interesting question is whether the PAOTR problem remains polynomial for AND-trees or whether it becomes NP-complete.

# References

[1] S. Gaonkar, J. Li, R. Roy Choudhury, L. Cox, and A. Schmidt. Micro-Blog: Sharing and Querying Content through Mobile Phones and Social Participation. In *Proc. of the ACM Intl. Conf. on Mobile Systems, Applications, and Services*, 2008.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[3] Russell Greiner, Ryan Hayward, Magdalena Jankowska, and Michael Molloy. Finding Optimal Satisficing Strategies for And-Or Trees. *Artificial Intelligence*, 170(1):19–58, 2006.

[4] L. Lim, A. Misra, and T. Mo. Adaptive Data Acquisition Strategies for Energy-Efficient Smartphone-based Continuous Processing of Sensor Streams. *Distributed Parallel Databases*, 31(2):321–351, 2013.

[5] E. Miluzzo. Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application. In *Proc. of ACM Conf. on Embedded Networked Sensor Systems*, 2008.

[6] I. Mohomed, A. Misra, M. Ebling, and W. Jerome. Context-Aware and Personalized Event Filtering for Low-Overhead Continuous Remote Health Monitoring. In *Proc. of the IEEE Intl. Symp. on a World of Wireless Mobile and Multimedia Networks*, 2008.

[7] The SHIMMER sensor platform. `http://shimmer-research.com`, 2013.

[8] D. E. Smith. Controlling backward inference. *Artificial Intelligence*, 39(2):145—208, 1989.

# A Proof of Proposition 1

*Proof.* We prove the proposition by contradiction. Consider an AND-tree and two leaves in the tree $l_1$ and $l_2$ that require data items from the same stream $S$ such that $d_1 > d_2$. We terms these leaves "inverted" because the earlier one, $l_1$, requires more data items than the later one, $l_2$. Assume that there is an optimal schedule $\xi$ in which $l_1$ is scheduled before $l_2$. Without loss of generality, we assume that $l_1$ is the first leaf in the schedule that is part of an inverted pair of leaves (if not, consider the earliest such leaf). Evaluating $l_2$ has always cost zero in this schedule because all data items required by $l_2$ are also required by $l_1$.

The sequence of leaves in $\xi$ can be written as: $l_{b_1}, \ldots, l_{b_t}, l_1, l_{m_1}, \ldots, l_{m_u}, l_2, l_{a_1}, \ldots, l_{a_v}$. The cost $\mathcal{C}$ of $\xi$ can be written:

$$\mathcal{C} = X + \mathcal{P}_b \cdot (d_1 - d_{LB})c(S) + \mathcal{P}_b \cdot p_1 \cdot Y + \mathcal{P}_b \cdot p_1 \cdot \mathcal{P}_m \cdot 0 + \mathcal{P}_b \cdot p_1 \cdot \mathcal{P}_m \cdot p_2 \cdot Z$$

where

- $\mathcal{P}_b = \prod_{i=1}^{t} p_{b_i}$ and $\mathcal{P}_m = \prod_{i=1}^{u} p_{m_i}$;
- $X$ is the expected cost of evaluating leaves $l_{b_1}, \ldots, l_{b_t}$ in that order;
- $Y$ is the expected cost of evaluating leaves $l_{m_1}, \ldots, l_{m_u}$ in that order if leaves $l_{b_1}, \ldots, l_{l_t}$ and $l_1$ all evaluate to TRUE;
- $Z$ is the expected cost of evaluating leaves $l_{a_1}, \ldots, l_{a_v}$ in that order if leaves $l_{b_1}, \ldots, l_{b_t}$, $l_1$, $l_{l_1}, \ldots, l_{m_u}$, and $l_2$ all evaluated to TRUE;
- $d_{LB} = \max_{i=1,\ldots,t}(d_{b_i})$, or the number of elements of stream $S$ that have been acquired after evaluating leaves $l_{b_1}, \ldots, l_{b_t}$.

Because $l_1$ and $l_2$ are the first two inverted leaves in $\xi$, $d_1 - d_{LB}$ is non-negative (otherwise a leaf among $l_{b_1}, \ldots, l_{b_t}$ and leaf $l_1$ would be inverted).

We now construct another schedule, $\xi$', as $l_{b_1}, \ldots, l_{b_t}, l_2, l_1, l_{m_1}, \ldots, l_{m_u}, l_{a_1}, \ldots, l_{a_v}$. The expected cost $\mathcal{C}$' of $\xi$' can then be written as:

$$\mathcal{C}' = X + \mathcal{P}_b \cdot (d_2 - d_{LB})c(S) + \mathcal{P}_b \cdot p_2(d_1 - d_2)c(S) + \mathcal{P}_b \cdot p_2 \cdot p_1 \cdot Y + \mathcal{P}_b \cdot p_2 \cdot p_1 \cdot \mathcal{P}_m \cdot Z$$

Because $l_1$ and $l_2$ are the first two inverted leaves in $\xi$, $d_2 - d_{LB}$ is non-negative (otherwise a leaf among $l_{b_1}, \ldots, l_{b_t}$ and leaf $l_2$ would be inverted). Computing the difference of the costs of both schedules yields:

$$\mathcal{C} - \mathcal{C}' \quad = \mathcal{P}_b(1 - p_2)\left((d_1 - d_2)c(S) + p_1 Y\right)$$

$\mathcal{C} - \mathcal{C}'$ is strictly positive because all costs are positives, all probabilities are between 0 and 1, and because $d_1 > d_2$ by assumption. This contradicts the optimality of $\xi$. □

# B Proof of Theorem 1

*Proof.* We prove the theorem by contradiction. We assume that there exists an instance for which the schedule produced by Algorithm 1, $\xi_{greedy}$, is not optimal. Among the optimal schedules, let us pick a schedule, $\xi_{opt}$, which has the longest prefix $\mathbb{P}$ in common with schedule $\xi_{greedy}$. We consider the first decision (i.e., one recursive call to the algorithm) taken by Algorithm 1 that schedules a leaf that does not belong to $\mathbb{P}$. Let $k$ be the number of leaves scheduled by this decision, and let us denote them $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$, scheduled in this order. Recall each call to the GREEDY algorithm schedules a sequence of leaves that all require data items from the same stream. Furthermore, the scheduled sequence of leaves is a sub-sequence of the ordered sequence of all leaves that require data items from that stream, sorted by increasing number of data items required. Without loss of generality, we assume that $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$ all require items from stream

1. The first of these leaves may belong to $\mathbb{P}$ (as the last leaf occurrences in $\mathbb{P}$). Let $\mathbb{P}'$ be equal to $\mathbb{P}$ minus the leaves $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$. Then, $\xi_{greedy}$ can be written as:

$$\xi_{greedy} = \mathbb{P}', l_{\sigma(1)}, ..., l_{\sigma(k)}, \mathbb{S}. \tag{2}$$

In turn, $\xi_{opt}$ can be written $\xi_{opt} = \mathbb{P}', \mathbb{Q}, \mathbb{R}$ where $l_{\sigma(k)}$ is the last leaf of $\mathbb{Q}$. In other words, $\mathbb{Q}$ can be written $L_1 l_{\sigma(1)} L_2 l_{\sigma(2)} ... L_k l_{\sigma(k)}$, where each sequence of leaves $L_i$, $1 \leq i \leq k$, can be empty. Note that, because of Theorem 1, and because the sequence $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$ is a sub-sequence of the the list of all leaves requiring data items from that stream sorted by increasing number of data items required, none of the $L_i$ sequences can contain a leaf requiring elements from stream 1. Therefore,

$$\xi_{opt} = \mathbb{P}', \mathbb{Q}, \mathbb{R} \text{ where } \mathbb{Q} = L_1 l_{\sigma(1)} L_2 l_{\sigma(2)} ... L_k l_{\sigma(k)} \tag{3}$$

From $\xi_{greedy}$ and $\xi_{opt}$, we build a new schedule, $\xi_{new}$, defined as

$$\xi_{new} = \mathbb{P}', NewOrder, \mathbb{R} \text{ where } NewOrder = l_{\sigma(1)}, ..., l_{\sigma(k)}, L_1, ..., L_k \tag{4}$$

$\mathbb{P}'$, $l_{\sigma(1)}$, ..., $l_{\sigma(k)}$ is a prefix to both $\xi_{greedy}$ and $\xi_{new}$. This prefix is strictly larger than $\mathbb{P}$ (since $\mathbb{P}$ does not contain $l_{\sigma(k)}$). Therefore, if the cost of $\xi_{new}$ is not greater than that of $\xi_{opt}$, $\xi_{new}$ is optimal and has a longer prefix in common with $\xi_{greedy}$ than $\xi_{new}$, which would contradict the definition of $\xi_{opt}$. We obtain this contradiction by computing the cost of $\xi_{new}$ and showing that it is no larger than that of $\xi_{opt}$.

**Cost notations** – To ease the writing of the proof we introduce several notations. If $\mathbb{X}$ is a partial leaf schedule, $P(\mathbb{X})$ denotes the probability that all leaves in $\mathbb{X}$ evaluates to TRUE. In other words, $P(\mathbb{X}) = \prod_{l_i \in \mathbb{X}} p_i$. Let $\mathbb{X}$ and $\mathbb{Y}$ be two disjoint (partial) leaf schedules, i.e., they do not have any leaf in common, such that $\mathbb{X}$ is evaluated right before $\mathbb{Y}$. Then $Cost(\mathbb{Y} \mid \mathbb{X})$ denotes the cost of evaluating $\mathbb{Y}$, assuming that all leaves in $\mathbb{X}$ have evaluated to TRUE. Of course, $Cost(\mathbb{Y} \mid \mathbb{X})$ takes into account all data items acquired during the successful evaluation of $\mathbb{X}$. With these notations, we can now give the costs of $\xi_{new}$ and $\xi_{opt}$ based on their definitions as sequences of partial leaf schedules in Equations (3) and (4):

$$\begin{aligned} Cost(\xi_{opt}) &= Cost(\mathbb{P}') + P(\mathbb{P}')Cost(\mathbb{Q} \mid \mathbb{P}') \\ &\quad + P(\mathbb{P}')P(\mathbb{Q})Cost(\mathbb{R} \mid \mathbb{P}', \mathbb{Q}) \\ Cost(\xi_{new}) &= Cost(\mathbb{P}') + P(\mathbb{P}')Cost(NewOrder \mid \mathbb{P}') \\ &\quad + P(\mathbb{P}')P(NewOrder)Cost(\mathbb{R} \mid \mathbb{P}', NewOrder) \end{aligned}$$

Because $\mathbb{Q}$ and $NewOrder$ contain exactly the same leaves, $P(\mathbb{Q}) = P(NewOrder)$ and $Cost(\mathbb{R} \mid \mathbb{P}', \mathbb{Q}) = Cost(\mathbb{R} \mid \mathbb{P}', NewOrder)$. Therefore,

$$Cost(\xi_{opt}) - Cost(\xi_{new}) = P(\mathbb{P}')\left(Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}')\right) \tag{5}$$

From what precedes, it now suffices to show that $Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}') \geq 0$ to prove the theorem.

**Initial mathematical formulation** – We use Proposition 1 to define notations that make it possible to obtain a simple expression for the quantity in Equation 5. Consider a stream $S$ and two leaves $l_i$ and $l_j$ that require, respectively, $d_i$ and $d_j$ items from stream $S$, with $d_i < d_j$. Then, according to Proposition 1, $l_i$ is always evaluated before $l_j$ in an optimal schedule. The GREEDY algorithm also schedules $l_i$ before $l_j$. If there does not exist any leaf $l_k$ requiring $d_k \in [d_i; d_j]$ elements from stream $s$, then each time $l_j$ is evaluated, exactly $d_j - d_i$ items are acquired from stream $s$, because the last $d_i$ elements of stream $s$ were acquired when $l_i$ was

evaluated. In this case, we define $a_i$ as the number of data items that must be acquired when evaluating leaf $l_i$. Formally,

$$a_i = d_i - \max\{d_j \mid S(j) = S(i) \text{ and } d_j < d_i\}$$

*Remark:* One should note that we can assume without loss of generality that the AND tree does not contain two leaves requiring the exact same number of items from the same stream. If such two leaves exist, then one replaces them by a single leaf with the same data item requirement and with a probability of success that is the product of the probability of success of the two original leaves. This is because once one of the two original leaves has been evaluated then the other one can be evaluated for free.

To ease the writing of the proof, we index the leaves in $L_1$, ..., $L_k$ according to the stream from which they require data items, and introduce the following additional notations. Let $N_i$ be the number of leaves in $L_1 \cup \ldots \cup L_k$ that require data items from stream $i$ and $l_{i,j}$ be the $j$-th of these leaves. We then extend the notations defined in Section 2 as follows: the probability of success of $l_{i,j}$ is $p_{i,j}$, $l_{i,j}$ requires $d_{i,j}$ elements from stream $S(i,j)$, etc. $\mu_{(i,j)}$ is the index of the leaf sequence $L_p$ to which leaf $l_{i,j}$ belongs: $l_{i,j} \in L_{\mu_{(i,j)}}$. $Q_{i,j}$ is the product of the success probabilities of the leaves that precede $l_{i,j}$ in $L_{\mu_{(i,j)}}$, $Q_m$ is the product of the success probabilities of all the leaves in $L_m$, and $\mathcal{Q}_m = \prod_{n=1}^{m} Q_n$. Finally, we define $P_m = \prod_{n=1}^{m} p_{\sigma(n)}$. With these notations we can now write $Cost(NewOrder \mid \mathbb{P}')$ as:

$$\begin{aligned} Cost(NewOrder \mid \mathbb{P}') &= \sum_{m=1}^{k} \left( \prod_{n=1}^{m-1} p_{\sigma(n)} \right) a_{\sigma(m)} \\ &\quad + \sum_{i=2}^{s} \sum_{j=1}^{N_i} \left( \prod_{m=1}^{k} p_{\sigma(m)} \right) \left( \prod_{m=1}^{\mu_{(i,j)}-1} Q_m \right) Q_{i,j} a_{i,j} c(S(i,j)) \\ &= \sum_{m=1}^{k} P_{m-1} a_{\sigma(m)} + \sum_{i=2}^{s} \sum_{j=1}^{N_i} P_k \mathcal{Q}_{\mu_{(i,j)}-1} Q_{i,j} a_{i,j} c(S(i,j)) \ , \end{aligned}$$

and $Cost(\mathbb{Q} \mid \mathbb{P}')$ as:

$$\begin{aligned} Cost(\mathbb{Q} \mid \mathbb{P}') &= \sum_{m=1}^{k} \left( \prod_{n=1}^{m-1} p_{\sigma(n)} \right) \left( \prod_{n=1}^{m} Q_n \right) a_{\sigma(m)} \\ &\quad + \sum_{i=2}^{s} \sum_{j=1}^{N_i} \left( \prod_{m=1}^{\mu_{(i,j)}-1} p_{\sigma(m)} \right) \left( \prod_{m=1}^{\mu_{(i,j)}-1} Q_m \right) Q_{i,j} a_{i,j} c(S(i,j)) \\ &= \sum_{m=1}^{k} P_{m-1} \mathcal{Q}_m a_{\sigma(m)} + \sum_{i=2}^{s} \sum_{j=1}^{N_i} P_{\mu_{(i,j)}-1} \mathcal{Q}_{\mu_{(i,j)}-1} Q_{i,j} a_{i,j} c(S(i,j)) \ . \end{aligned}$$

Therefore:

$$\begin{aligned}
Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}') &= \sum_{m=1}^{k} P_{m-1}\mathcal{Q}_m a_{\sigma(m)} \\
&+ \sum_{i=2}^{s}\sum_{j=1}^{N_i} P_{\mu_{(i,j)}-1}\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}a_{i,j}c(S(i,j)) \\
&- \left(\sum_{m=1}^{k} P_{m-1}a_{\sigma(m)} + \sum_{i=2}^{s}\sum_{j=1}^{N_i} P_k\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}a_{i,j}c(S(i,j))\right) \\
&= \sum_{m=1}^{k} P_{m-1}(\mathcal{Q}_m - 1)a_{\sigma(m)} \\
&+ \sum_{i=2}^{s}\sum_{j=1}^{N_i} \left(P_{\mu_{(i,j)}-1} - P_k\right)\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}a_{i,j}c(S(i,j)) \ .
\end{aligned}$$

We introduce two additional notations:

$$\alpha_{(i,j)} = \mathcal{Q}_{\mu_{(i,j)}-1}\left(P_{\mu_{(i,j)}-1} - P_k\right)Q_{i,j} \ , \text{and}$$

$$A = \frac{\sum_{m=1}^{k} P_{m-1}a_{\sigma(m)}}{1-P_k} \ ,$$

so that we can finally write the expression for the difference of the two costs:

$$Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}') = \left(\sum_{m=1}^{k} P_{m-1}\left(\mathcal{Q}_m - 1\right)a_{\sigma(m)}\right) + \left(\sum_{i=2}^{s}\sum_{j=1}^{N_i}\alpha_{(i,j)}a_{i,j}c(S(i,j))\right) \ . \tag{6}$$

**Accounting for the algorithm's scheduling decisions** – The best decision for the GREEDY algorithm was to evaluate at once the leaf sequence $l_{\sigma(1)}, ..., l_{\sigma(k)}$. Therefore, as far as the algorithm is concerned, this was a better decision than evaluating any sequence of leaves from any other stream. More formally, for any stream $i$, $2 \leq i \leq s$, and the set of the first $j$ leaves of that stream, $1 \leq j \leq N_i$, we have:

$$\frac{\left(\sum_{m=1}^{k}\left(\prod_{n=1}^{m-1} p_{\sigma(n)}\right)a_{\sigma(m)}\right)}{\left(1-\prod_{m=1}^{k} p_{\sigma(m)}\right)}\left(1-\prod_{l=1}^{j} p_{i,l}\right) \leq \left(\sum_{l=1}^{j}\left(\prod_{r=1}^{l-1} p_{i,r}\right)a_{i,l}c(S(i,l))\right) \ .$$

These equations express the fact that these other sequence of leaves of a *Ratio* value (see Algorithm 1) lower than that of the sequence scheduled by the algorithm, and can be rewritten as:

$$\textsc{Ineq}(i,j): \qquad A\left(1-\prod_{l=1}^{j} p_{i,l}\right) \leq \left(\sum_{l=1}^{j}\left(\prod_{r=1}^{l-1} p_{i,r}\right)a_{i,l}c(S(i,l))\right) \ . \tag{7}$$

**Determining multiplying coefficients** – To prove the theorem we combine the Inequalities (7) obtained for different values of $i$ and $j$. The idea is to follow a variable elimination process. $\textsc{Ineq}(i, N_i)$ is the only inequality in which $a_{i,N_i}$ appears. We multiply $\textsc{Ineq}(i, N_i)$ by a value $\lambda_{i,N_i}$ such that, in the resulting inequality, the coefficient of $a_{i,N_i}$ is the same than in

Equation (6). Next, we multiply $\textsc{Ineq}(i, N_i - 1)$ by a value $\lambda_{i,N_i-1}$ such that when adding the resulting inequality to the one previously obtained, the coefficient of $a_{i,N_i-1}$ is the same than in Equation (6), and so on. This process can be done independently for the different streams as $\textsc{Ineq}(i, j)$ only contains terms relative to stream $i$.

We define the $\lambda_{i,j}$'s are defined as follows.

$$\lambda_{i,j} = \begin{cases} \dfrac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} & \text{if } j = N_i \ , \\ \dfrac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} - \dfrac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} & \text{otherwise.} \end{cases}$$

We will later show that this choice of multipliers enable us to achieve our goal. However, as we want to use the $\lambda_{i,j}$'s as multiplying coefficients for inequalities, we must first show that they are all non-negative. This is evident for the $\lambda_{i,N_i}$'s. Let us consider $\lambda_{i,j}$ for $j \in [1; N_i - 1]$:

$$\begin{aligned} \lambda_{i,j} &= \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} - \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \\ &= \frac{1}{\prod_{l=1}^{j-1} p_{i,l}} \left( \prod_{m=1}^{\mu_{(i,j)}-1} Q_m \right) \left( \prod_{m=1}^{\mu_{(i,j)}-1} p_{\sigma(m)} \right) \left( 1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)} \right) Q_{i,j} \\ &\quad - \frac{1}{\prod_{l=1}^{j} p_{i,l}} \left( \prod_{m=1}^{\mu_{(i,j+1)}-1} Q_m \right) \left( \prod_{m=1}^{\mu_{(i,j+1)}-1} p_{\sigma(m)} \right) \left( 1 - \prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)} \right) Q_{i,j+1} \\ &= \frac{1}{\prod_{l=1}^{j-1} p_{i,l}} \left( \prod_{m=1}^{\mu_{(i,j)}-1} Q_m p_{\sigma(m)} \right) \\ &\quad \times \left[ \left( 1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)} \right) Q_{i,j} - \left( \prod_{m=\mu_{(i,j)}}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)} \right) \left( 1 - \prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)} \right) \frac{Q_{i,j+1}}{p_{i,j}} \right] \end{aligned}$$

Let us first consider the case $\mu_{(i,j+1)} = \mu_{(i,j)}$. Then the above equation can be rewritten:

$$\lambda_{i,j} = \frac{1}{\prod_{l=1}^{j-1} p_{i,l}} \left( \prod_{m=1}^{\mu_{(i,j)}-1} Q_m p_{\sigma(m)} \right) \left( 1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)} \right) \left[ Q_{i,j} - \frac{Q_{i,j+1}}{p_{i,j}} \right]$$

By definition, $Q_{i,j+1}$ is the product of the probabilities of success of all the leaves that are evaluated before the leaf $l_{i,j+1}$ is evaluated. By definition of the numbering of the leaves, this includes at least all the leaves that are evaluated before leaf $l_{i,j}$ is evaluated and leaf $l_{i,j}$. As all probabilities are less than or equal to 1, this implies that $Q_{i,j+1} \leq Q_{i,j} p_{i,j}$, and therefore that $\lambda_{i,j} \geq 0$.

We now consider the other case: $\mu_{(i,j+1)} > \mu_{(i,j)}$. Then, $Q_{\mu_{(i,j)}}$ is of the form $Q_{i,j} p_{i,j} X$ where $X$ is the product of the probabilities of success of the leaves appearing in $L_{\mu_{(i,j)}}$ after the leaf $l_{i,j}$. Therefore, $Q_{i,j} p_{i,j} \geq Q_{\mu_{(i,j)}}$. As for all $i \in [1; s]$, $0 \leq p_{\sigma(i)} \leq 1$, $\prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)} \geq \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}$ and

$1 - \prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)} \leq 1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}$, $p_{\sigma(\mu_{(i,j)})}(\prod_{m=\mu_{(i,j)}+1}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)})Q_{i,j+1} \leq 1$ because it is a product of probabilities. Using these inequalities, we have:

$$\begin{aligned}
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right) Q_{i,j} - \left(\prod_{m=\mu_{(i,j)}}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)}\right)\left(1 - \prod_{m=\mu_{(i,j+1)}}^{k} p_{\sigma(m)}\right)\frac{Q_{i,j+1}}{p_{i,j}} &\geq \\
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right) Q_{i,j} - \left(\prod_{m=\mu_{(i,j)}}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)}\right)\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right)\frac{Q_{i,j+1}}{p_{i,j}} &= \\
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right)\left(Q_{i,j} - \left(\prod_{m=\mu_{(i,j)}}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)}\right)\frac{Q_{i,j+1}}{p_{i,j}}\right) &\geq \\
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right)\left(Q_{i,j} - Q_{\mu_{(i,j)}}\left(p_{\sigma(\mu_{(i,j)})}(\prod_{m=\mu_{(i,j)}+1}^{\mu_{(i,j+1)}-1} Q_m p_{\sigma(m)})Q_{i,j+1}\right)\frac{1}{p_{i,j}}\right) &\geq \\
&\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right)\left(Q_{i,j} - Q_{\mu_{(i,j)}}\frac{1}{p_{i,j}}\right) &\geq 0
\end{aligned}$$

Therefore, all the $\lambda_{i,j}$'s are non-negative.

**Combining the inequalities** – For a given couple of values $(i, j)$, with $2 \leq i \leq s$ and $1 \leq j \leq N_i$, let $\textsc{Ineq}(i, j)$ be Inequality (7) defined for $(i, j)$. Because all the $\lambda_{i,j}$'s are non-negative, we can form the inequality:

$$\sum_{i=2}^{s}\sum_{j=1}^{N_i}(\lambda_{i,j} \times \textsc{Ineq}(i,j)) \tag{8}$$

We now show that Inequality (8) leads to:

$$Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}') \geq \sum_{m=1}^{k} P_{m-1}\left(\mathcal{Q}_m - 1\right) a_{\sigma(m)} + A\sum_{i=2}^{s}\left(\sum_{j=1}^{N_i}\alpha_{(i,j)}\left(1 - p_{i,j}\right)\right) \tag{9}$$

To prove the Inequality (9), we consider the terms relative to stream $i$ in Inequality (8):

$$\begin{aligned}
&\sum_{j=1}^{N_i}(\lambda_{i,j} \times \textsc{Ineq}(i,j)) \qquad \Leftrightarrow \\
A\sum_{j=1}^{N_i}\lambda_{i,j}\left(1 - \prod_{l=1}^{j} p_{i,l}\right) \quad &\leq \quad \sum_{j=1}^{N_i}\lambda_{i,j}\left(\sum_{l=1}^{j}\left(\prod_{r=1}^{l-1} p_{i,r}\right) a_{i,l}\, c(S(i,l))\right)
\end{aligned} \tag{10}$$

We start by considering the left-hand side of this inequality.

$$
\begin{aligned}
&\sum_{j=1}^{N_i} \lambda_{i,j} \left(1 - \prod_{l=1}^{j} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i-1} \lambda_{i,j} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) + \lambda_{i,N_i} \left(1 - \prod_{l=1}^{N_i} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i-1} \left(\frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} - \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}}\right) \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left(1 - \prod_{l=1}^{N_i} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) - \left(\sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left(1 - \prod_{l=1}^{N_i} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) - \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j-1} p_{i,l}\right)\right) + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left(1 - \prod_{l=1}^{N_i} p_{i,l}\right) && = \\
&\left(\sum_{j=1}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) - \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j-1} p_{i,l}\right)\right) && = \\
&\alpha_{(i,1)} \left(1 - p_{i,1}\right) + \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j} p_{i,l}\right)\right) - \left(\sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(1 - \prod_{l=1}^{j-1} p_{i,l}\right)\right) && = \\
&\alpha_{(i,1)} \left(1 - p_{i,1}\right) + \sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left(\prod_{l=1}^{j-1} p_{i,l} - \prod_{l=1}^{j} p_{i,l}\right) && = \\
&\alpha_{(i,1)} \left(1 - p_{i,1}\right) + \sum_{j=2}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right) && = \\
&\sum_{j=1}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right) \ . &&
\end{aligned}
$$

Therefore,

$$
A \sum_{j=1}^{N_i} \lambda_{i,j} \left(1 - \prod_{l=1}^{j} p_{i,l}\right) = A \left(\sum_{j=1}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right)\right) \ . \tag{11}
$$

We now focus on the right-hand side of the inequality:

$$
\begin{aligned}
&\sum_{j=1}^{N_i} \lambda_{i,j} \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) = \\
&\left( \sum_{j=1}^{N_i-1} \lambda_{i,j} \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) + \lambda_{i,N_i} \left( \sum_{l=1}^{N_i} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) = \\
&\left( \sum_{j=1}^{N_i-1} \left( \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} - \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \right) \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) \\
&\quad + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left( \sum_{l=1}^{N_i} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) = \\
&\left( \sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) - \left( \sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) \\
&\quad + \frac{\alpha_{(i,N_i)}}{\prod_{l=1}^{N_i-1} p_{i,l}} \left( \sum_{l=1}^{N_i} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) = \\
&\left( \sum_{j=1}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) - \left( \sum_{j=1}^{N_i-1} \frac{\alpha_{(i,j+1)}}{\prod_{l=1}^{j} p_{i,l}} \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) = \\
&\left( \sum_{j=1}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) - \left( \sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left( \sum_{l=1}^{j-1} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) = \\
&\alpha_{(i,1)} a_{i,1} c(S(i,1)) + \left( \sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) \\
&\quad - \left( \sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left( \sum_{l=1}^{j-1} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) \right) = \\
&\alpha_{(i,1)} a_{i,1} c(S(i,1)) + \sum_{j=2}^{N_i} \frac{\alpha_{(i,j)}}{\prod_{l=1}^{j-1} p_{i,l}} \left( \prod_{r=1}^{j-1} p_{i,r} \right) a_{i,j} c(S(i,j)) = \\
&\sum_{j=1}^{N_i} \alpha_{(i,j)} a_{i,j} c(S(i,j)) \; .
\end{aligned}
$$

Therefore

$$
\sum_{j=1}^{N_i} \lambda_{i,j} \left( \sum_{l=1}^{j} \left( \prod_{r=1}^{l-1} p_{i,r} \right) a_{i,l} c(S(i,l)) \right) = \left( \sum_{j=1}^{N_i} \alpha_{(i,j)} a_{i,j} c(S(i,j)) \right) \; . \tag{12}
$$

By combining Inequality (10) with Equations (11) and (12), and by summing over all streams, we obtain:

$$
A \sum_{i=2}^{s} \left( \sum_{j=1}^{N_i} \alpha_{(i,j)} \left(1 - p_{i,j}\right) \right) \leq \sum_{i=2}^{s} \left( \sum_{j=1}^{N_i} \alpha_{(i,j)} a_{i,j} c(S(i,j)) \right) \; . \tag{13}
$$

Using Equation (6) and Inequality (13), we obtain:

$$Cost(\mathbb{Q} \mid \mathbb{P}') - Cost(NewOrder \mid \mathbb{P}') \geq \sum_{m=1}^{k} P_{m-1}\,(\mathcal{Q}_m - 1)\,a_{\sigma(m)} + A \sum_{i=2}^{s} \left( \sum_{j=1}^{N_i} \alpha_{(i,j)}\,(1 - p_{i,j}) \right) . \tag{14}$$

**Completing the proof** – We want to prove that the right-hand side of Inequality (14) is non-negative, i.e., that the following inequality holds:

$$\sum_{m=1}^{k} P_{m-1}\,(\mathcal{Q}_m - 1)\,a_{\sigma(m)} + A \sum_{i=2}^{s} \left( \sum_{j=1}^{N_i} \alpha_{(i,j)}\,(1 - p_{i,j}) \right) \geq 0 . \tag{15}$$

Because of Inequality (14), this will enable us to conclude. Let

$$A_n = \sum_{m=1}^{n} P_{m-1} a_{\sigma(m)} .$$

Therefore, $A = \frac{A_k}{(1-P_k)}$. We start by focusing on the first term of Inequality (15). We prove that:

$$\sum_{m=1}^{k} P_{m-1}\,(\mathcal{Q}_m - 1)\,a_{\sigma(m)} \geq A \left( \left( \sum_{i=1}^{k} (\mathcal{Q}_i - \mathcal{Q}_{i-1}) P_{i-1} \right) + P_k (1 - \mathcal{Q}_k) \right) . \tag{16}$$

$$
\begin{aligned}
&\sum_{m=1}^{k} P_{m-1}\,(\mathcal{Q}_m - 1)\,a_{\sigma(m)} \\
&= \sum_{m=1}^{k} (\mathcal{Q}_m - 1) \left( \sum_{n=1}^{m} P_{n-1} a_n - \sum_{n=1}^{m-1} P_{n-1} a_n \right) \\
&= \sum_{m=1}^{k} (\mathcal{Q}_m - 1)(A_m - A_{m-1}) \\
&= \left( \sum_{m=1}^{k} \mathcal{Q}_m (A_m - A_{m-1}) \right) - \left( \sum_{m=1}^{k} (A_m - A_{m-1}) \right) \\
&= \left( \sum_{m=1}^{k} (\mathcal{Q}_m A_m - \mathcal{Q}_m A_{m-1}) \right) - (A_k - A_0) \\
&= \left( \sum_{m=1}^{k} (\mathcal{Q}_m A_m - \mathcal{Q}_{m-1} A_{m-1} + \mathcal{Q}_{m-1} A_{m-1} - \mathcal{Q}_m A_{m-1}) \right) - A_k \\
&= \left( \sum_{m=1}^{k} (\mathcal{Q}_m A_m - \mathcal{Q}_{m-1} A_{m-1}) \right) + \left( \sum_{m=1}^{k} (\mathcal{Q}_{m-1} A_{m-1} - \mathcal{Q}_m A_{m-1}) \right) - A_k \\
&= (\mathcal{Q}_k A_k - \mathcal{Q}_0 A_0) + \left( \sum_{m=1}^{k} (\mathcal{Q}_{m-1} - \mathcal{Q}_m) A_{m-1} \right) - A_k \\
&= \left( \sum_{m=1}^{k} (\mathcal{Q}_{m-1} - \mathcal{Q}_m) A_{m-1} \right) + (\mathcal{Q}_k - 1) A_k = \left( \sum_{m=1}^{k} (\mathcal{Q}_{m-1} - \mathcal{Q}_m) A_{m-1} \right) + (\mathcal{Q}_k - 1) A (1 - P_k) \; .
\end{aligned}
$$

By hypothesis, the best decision for the greedy algorithm was to read at once the leaves $a_1$, ..., $a_k$. Therefore, this was better than reading any other sequence of leaves from stream 1. So, for any value of $m \leq k$,

$$A\,(1 - P_{m-1}) \leq A_{m-1} \; .$$

Because $\mathcal{Q}_m = \mathcal{Q}_{m-1}Q_m$ with $Q_m \in [0;1]$, then $\mathcal{Q}_{m-1} - \mathcal{Q}_m \geq 0$. Therefore, we have:

$$\begin{aligned}
&\sum_{m=1}^{k}(\mathcal{Q}_{m-1} - \mathcal{Q}_m)A_{m-1} + A(1-P_k)(\mathcal{Q}_k - 1) \\
&\geq A\left[\left(\sum_{m=1}^{k}(\mathcal{Q}_{m-1} - \mathcal{Q}_m)(1-P_{m-1})\right) + (1-P_k)(\mathcal{Q}_k - 1)\right] \\
&= A\left[\left(\sum_{m=1}^{k}(\mathcal{Q}_{m-1} - \mathcal{Q}_m)\right) - \left(\sum_{m=1}^{k}(\mathcal{Q}_{m-1} - \mathcal{Q}_m)P_{m-1}\right) + (1-P_k)(\mathcal{Q}_k - 1)\right] \\
&= A\left[(\mathcal{Q}_0 - \mathcal{Q}_k) + \left(\sum_{m=1}^{k}(\mathcal{Q}_m - \mathcal{Q}_{m-1})P_{m-1}\right) + (1-P_k)(\mathcal{Q}_k - 1)\right] \\
&= A\left[1 - \mathcal{Q}_k + \left(\sum_{m=1}^{k}(\mathcal{Q}_m - \mathcal{Q}_{m-1})P_{m-1}\right) + (1-P_k)(\mathcal{Q}_k - 1)\right] \\
&= A\left[\left(\sum_{m=1}^{k}(\mathcal{Q}_m - \mathcal{Q}_{m-1})P_{m-1}\right) - P_k(\mathcal{Q}_k - 1)\right] .
\end{aligned}$$

We now focus on the second term of Inequality (15). We prove must prove that:

$$\sum_{i=2}^{s}\sum_{j=1}^{N_i}\alpha_{(i,j)}(1-p_{i,j}) = \left(\sum_{m=1}^{k}\mathcal{Q}_{m-1}P_{m-1}(1-Q_m)\right) - P_k(1-\mathcal{Q}_k) . \tag{17}$$

We have:

$$\begin{aligned}
&\sum_{i=2}^{s}\sum_{j=1}^{N_i}\alpha_{(i,j)}(1-p_{i,j}) \\
&= \sum_{i=2}^{s}\sum_{j=1}^{N_i}\left(\prod_{m=1}^{\mu_{(i,j)}-1}Q_m p_{\sigma(m)}\right)\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right)Q_{i,j}(1-p_{i,j}) \\
&= \sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}\left(1 - \prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right)Q_{i,j}(1-p_{i,j}) \\
&= \left(\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - \left(\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}\left(\prod_{m=\mu_{(i,j)}}^{k} p_{\sigma(m)}\right)Q_{i,j}(1-p_{i,j})\right) \\
&= \left(\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - \left(\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_k Q_{i,j}(1-p_{i,j})\right) \\
&= \left(\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - \left(P_k\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) .
\end{aligned}$$

We concentrate on the second term and its meaning. For any stream $i$ and any of its leaves $j$, $\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}$ is the probability of success of all the leaves evaluated before the studied leaf (not

considering the leaves of other streams), and $\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}p_{i,j}$ is the same probability right after the evaluation of the studied leaf. Therefore, in the inner sum all terms cancel out except the first one, that is the probability of success if no leaf had been evaluated so far, and the last one, that is the probability of success if all the leaves have been evaluated. Formally, we have:

$$\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j}) = 1 - \left(\prod_{m=1}^{k} Q_m\right) = 1 - \mathcal{Q}_k \ . \tag{18}$$

Therefore,

$$\begin{aligned}
&\sum_{i=2}^{s}\sum_{j=1}^{N_i}\alpha_{(i,j)}(1-p_{i,j}) \\
&= \left(\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - \left(P_k\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) \\
&= \left(\sum_{i=2}^{s}\sum_{j=1}^{N_i}\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}(1-p_{i,j})\right) - P_k(1-\mathcal{Q}_k) \\
&= \left(\sum_{i=2}^{s}\sum_{j=1}^{N_i}(\mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j} - \mathcal{Q}_{\mu_{(i,j)}-1}P_{\mu_{(i,j)}-1}Q_{i,j}p_{i,j})\right) - P_k(1-\mathcal{Q}_k) \\
&= \left(\sum_{m=1}^{k}\sum_{\substack{(i,j)\ \text{s.t.}\\ \mu_{(i,j)}=m}}(\mathcal{Q}_{m-1}P_{m-1}Q_{i,j} - \mathcal{Q}_{m-1}P_{m-1}Q_{i,j}p_{i,j})\right) - P_k(1-\mathcal{Q}_k) \\
&= \left(\sum_{m=1}^{k}\mathcal{Q}_{m-1}P_{m-1}\sum_{\substack{(i,j)\ \text{s.t.}\\ \mu_{(i,j)}=m}}(Q_{i,j} - Q_{i,j}p_{i,j})\right) - P_k(1-\mathcal{Q}_k) \\
&= \left(\sum_{m=1}^{k}\mathcal{Q}_{m-1}P_{m-1}(1-Q_m)\right) - P_k(1-\mathcal{Q}_k) \ .
\end{aligned}$$

The last equality above is established using the same type of reasoning as the one we used to establish Equation (18).

We now combine Inequality (16) with Equation (17):

$$\begin{aligned}
&\sum_{m=1}^{k} (\mathcal{Q}_m - 1)\, P_{m-1} a_{\sigma(m)} + A \sum_{i=2}^{s} \sum_{j=1}^{N_i} \alpha_{(i,j)}(1 - p_{i,j}) \\
&\geq A\left(\left(\sum_{m=1}^{k} (\mathcal{Q}_m - \mathcal{Q}_{m-1}) P_{m-1}\right) + P_k(1 - \mathcal{Q}_k)\right) + A\left(\left(\sum_{m=1}^{k} \mathcal{Q}_{m-1} P_{m-1}(1 - Q_m)\right) - P_k(1 - \mathcal{Q}_k)\right) \\
&= A\left(\left(\sum_{m=1}^{k} (\mathcal{Q}_m - \mathcal{Q}_{m-1}) P_{m-1}\right) + P_k(1 - \mathcal{Q}_k) + \left(\sum_{m=1}^{k} \mathcal{Q}_{m-1} P_{m-1}(1 - Q_m)\right) - P_k(1 - \mathcal{Q}_k)\right) \\
&= A\left(\left(\sum_{m=1}^{k} (\mathcal{Q}_m - \mathcal{Q}_{m-1}) P_{m-1}\right) + \left(\sum_{m=1}^{k} \mathcal{Q}_{m-1} P_{m-1}(1 - Q_m)\right)\right) \\
&= A\left(\sum_{m=1}^{k} (\mathcal{Q}_m P_{m-1} - \mathcal{Q}_{m-1} P_{m-1} + \mathcal{Q}_{m-1} P_{m-1} - \mathcal{Q}_{m-1} P_{m-1} Q_m)\right) \\
&= 0\ ,
\end{aligned}$$

because $\mathcal{Q}_{m-1} Q_m = \mathcal{Q}_m$. We have thus established Inequality (15), which concludes the proof. □

# C Proof of Proposition 2

*Proof.* Consider a schedule $\xi$, and a leaf in that schedule, $l_{i,j}$, which requires the $t$-th data item from stream $S_k$ (i.e., $S(i,j) = S_k$). Let us prove the first part of the proposition. If a leaf $l_{i,r}$ (i.e., a leaf under the same AND node as $l_{i,j}$) occurs before $l_{i,j}$ in $\xi$ and requires the $t$-th item from stream $S_k$ (i.e., $l_{i,r} \in \mathcal{L}_{k,t}$), then there are two possibilities. Either $l_{i,r}$ has been evaluated, in which case the evaluation of $l_{i,j}$ uses a data item that has already been acquired previously, hence a cost of 0. Or $l_{i,k}$ has not been evaluated, meaning that its evaluation was shortcircuited. In this case the AND node has evaluated to FALSE and the evaluation of $l_{i,j}$ is also shortcircuited, hence a cost of 0.

The second part of the proposition shows the expected cost as a product of three factors, each of which is a probability, and a fourth factor, $c(S(i,j))$, which is the cost of acquiring the data item from the stream. The interpretation of the expression for $\mathcal{C}_{i,j,t}$ is as follows: a leaf must acquire the item if and only if (i) the item has not been previously acquired; and (ii) no AND node has already evaluated to TRUE; and (iii) no leaf in the same AND node has already evaluated to FALSE. We explain the computation of these three probabilities hereafter.

The first factor is the probability that none of the leaves that precede $l_{i,j}$ in $\xi$ and that require the $t$-th item from stream $S_k$ have been evaluated. Such a leaf $l_{r,s}$ is evaluated if all the leaves in the same AND node that precede it in the schedule have evaluated to TRUE, which happens with probability $\prod_{l_{r,u} \prec l_{r,s}} p_{r,u}$, hence the expression for the first factor.

The second factor is the probability that none of the AND nodes that have been fully evaluated so far has evaluated to TRUE, since if this were the case the evaluation of $l_{i,j}$ would not be needed, leading to a cost of 0. Given an AND node in $\mathcal{A}_{i,j}$, say the $k$-th AND node, the probability that it has been evaluated to TRUE is $\prod_{r=1}^{m_k} p_{k,r}$. This is true except if one of the leaves of that AND node belongs to $\mathcal{L}_{k,t}$. The first factor assumes that that leaf was not

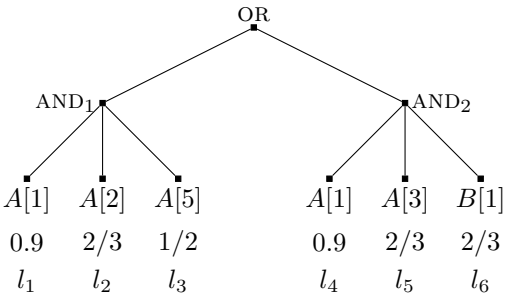



Figure 7: Example showing that Algorithm 1 does not optimally schedule the leaves of a DNF tree.

evaluated and, therefore, that that entire AND node was not evaluated. Hence, the expression for the second factor.

The third factor is the probability that all the leaves of the same AND node as the AND node of $l_{i,j}$ have evaluated to TRUE. Recall that, because we are in the second case of the proposition, none of these leaves requires the $t$-th item of stream $S_k$. All these leaves must evaluate to TRUE, otherwise the evaluation of $l_{i,j}$ would be shortcircuited, for a cost of 0, hence the expression for the third factor. □

# D Counter-example to the optimality of Algorithm 1 for DNF trees

In the *read-once* case, the following greedy algorithm [3] is optimal to evaluate a DNF tree:

- Each AND node ANDi is evaluated independently, sorting the leaves $l_{i,j}$ by increasing $d_{i,j}c(S(i,j))/q_{i,j}$. This leads to a cost $\mathcal{C}$i for node ANDi;
- Node ANDi is replaced by a single leaf node $l_i$ of cost $\mathcal{C}$i and success probability $p_i = \prod_{j=1}^{m_i} p_{i,j}$;
- These leaf nodes are scheduled by increasing $\mathcal{C}i/p_i$.

In this section we provide a counterexample to show that in the *shared* case, Algorithm 1 (optimal to evaluate a AND tree) cannot the used to evaluate a DNF tree.

We consider the example of Figure 7 where both streams have a cost of 1. There are two possible orders for evaluating the AND nodes. We consider both of them and explicit the behavior of Algorithm 1 on each of the AND's.

- AND$_1$ then AND$_2$. Because of Proposition 1, the leaves of AND$_1$ are always evaluated in the order $l_1$, $l_2$, $l_3$. The cost of evaluation of AND$_1$ is then $\alpha_1 = 1 + 0.9 \times (1 + 2/3 \times 3) = 3.7$. We then move to the evaluation of AND$_2$. Algorithm 1 first schedules leaf $l_5$ whose cost is null. We then have to compare the ratios for leaves $l_5$ and $l_6$:
  - $l_5$. The first element of $A$ was acquired for the evaluation of leaf $l_1$. The second element of $A$ needs to be acquired for $l_2$ only if leaf $l_2$ was not evaluated and leaf $l_4$ evaluated to TRUE, which happens with probability $(1 - 0.9) \times 0.9 = 0.09$. The third element of $A$ needs to be acquired only if leaf $l_3$ was not evaluated and leaf $l_4$ evaluated to TRUE, which happens with probability $(1 - 0.9 \times 2/3) \times 0.9 = 0.36$. Therefore, the evaluation cost of $l_5$ is $0 + 0.09 \times 1 + 0.36 \times 1 = 0.45$. The ratio for leaf $l_5$ is thus $\frac{0.45}{1-2/3} = 1.35$.

– $l_6$. The ratio for $l_6$ is $\frac{1}{1-2/3} = 3$.

Therefore, Algorithm 1 schedules $l_5$ and then $l_6$ for the overall cost:

$$3.7 + 0 + 0.45 + (1 - 0.9 \times 2/3 \times 1/2) \times 0.9 \times 2/3 = 4.57.$$

- $\text{AND}_2$ then $\text{AND}_1$. We first consider the ratios for the three leaves:
  – The ratio for $l_4$ is $\frac{1}{1-0.9} = 10$.
  – The ratio for $l_5$ is $\frac{1+0.9\times2}{1-0.9\times2/3} = 7$.
  – The ratio for $l_6$ is $\frac{1}{1-2/3} = 3$.

  Therefore the first scheduled leaf if $l_6$. Then the overall schedule is $l_6$, $l_4$, $l_5$, $l_1$, $l_2$, $l_3$. We compute the evaluation cost of each leaf.
  – $l_6$. Its cost is 1.
  – $l_4$. Its cost is $2/3 \times 1$.
  – $l_5$. Its cost is $2/3 \times 0.9 \times 2 = 1.2$.
  – $l_1$. Its cost is $(1 - 2/3) \times 1$.
  – $l_2$. Its costs is $(1 - 2/3 \times 0.9) \times 0.9 = 0.36$.
  – $l_3$. Its costs is $(1 - 2/3 \times 0.9) \times 1 + (1 - 2/3 \times 0.9 \times 2/3) \times 0.9 \times 2/3 \times 2 = 0.96$.

  The overall cost is thus 4.52.

We now consider the schedule $l_4$, $l_5$, $l_6$, $l_1$, $l_2$, $l_3$. We compute the evaluation cost of each leaf.

- The cost of $l_4$ is 1.
- The cost of $l_5$ is $0.9 \times 2 = 1.8$.
- The cost of $l_6$ is $0.9 \times 2/3 \times 1 = 0.6$.
- The cost of $l_1$ is 0.
- The cost of $l_2$ is $(1 - 0.9) \times 0.9 = 0.09$.
- The cost of $l_3$ is $(1 - 0.9) \times 0.9 \times 2/3 + (1 - 0.9 \times 2/3 \times 2/3) \times 0.9 \times 2/3 \times 2 = 0.78$.

The overall cost of this schedule is thus 4.27. The intuitive explanation is the following: for AND2 alone, the best evaluation order is $l_6$, $l_4$, $l_5$ (and this is the order chosen by Algorithm 1). However, because of the re-use of some data items of stream $A$ in AND1, the optimal order for the whole DNF tree is not the same! This leads to a huge combinatorial search space for the optimal ordering, which corroborates the hardness result (NP-completeness stated in Theorem 3) of the evaluation of DNF trees in the *shared* case.

# E Proof of Theorem 3

*Proof.* The problem is obviously in NP: given a schedule, i.e., an ordering of the leaves, one can compute its expected cost in polynomial time, using the method given in Section 4.1. The NP-completeness is obtained by reduction from 2-PARTITION [2]. Let $\mathcal{I}_1$ be an instance from 2-PARTITION: given a set $\{a_1, ..., a_n\}$ and $S = \sum_{i=1}^{n} a_i$, does there exist a subset $I$ such that $\sum_{i \in I} a_i = \frac{S}{2}$? We assume that $S$ is even, otherwise there is no solution. The size of $\mathcal{I}_1$ is $O(n+\log M)$, where $M = \max_{1 \le i \le n}\{a_i\}$. Without loss of generality, we assume that $M \ge 10$.We construct the following instance $\mathcal{I}_2$ of DNF-Decision:

- We consider a DNF tree with $N = n + 1$ AND nodes $\text{AND}_i$, $1 \le i \le n + 1$ and a total of $m = 2n + 1$ leaves.
- The set of streams is $\mathcal{S} = \{A_1, \ldots, A_n, B\}$. The cost of stream $S_i = A_i$ for $i \le n$ is $c(i) = \frac{1}{2Z}$, where $Z$ is some large constant defined below. The cost of stream $S_{n+1} = B$ is $c(n + 1) = C_0$, where $C_0 \approx \frac{1}{2}$ is a constant defined below.

- Each $\text{AND}_i$ node, where $i \leq n$, has a single leaf $l_{i,1}$ which has success probability

$$p_{i,1} = \frac{a_i}{Z} + \beta \frac{a_i^2}{Z^2}$$

  where $\beta \approx \frac{1}{2}$ is a constant defined below, and which requires $d_{i,1} = 2a_i$ elements of stream $S(i,1) = A_i$. Hence the cost to access all items of leaf $l_{i,1}$ is $d_{i,1}c(i) = \frac{a_i}{Z}$.
- The last AND node $AND_{n+1}$ has $m_{n+1} = n+1$ leaves which are specified as follows:
  - Each leaf $l_{n+1,i}$, where $i \leq n$, has success probability $p_{n+1,i} = 1 - \varepsilon$ and requires $d_{n+1,i} = a_i$ elements of stream $S(n+1,i) = A_i$. Hence the cost to access all items of leaf $l_{n+1,i}$ is $\frac{a_i}{2Z}$.
  - The last leaf $l_{n+1,n+1}$ has success probability $p_{n+1,n+1} = 1-\varepsilon$ and requires $d_{n+1,n+1} = 1$ element of stream $S(n+1,n+1) = B$ (at cost $c(n+1) = C_0$)
  - The constant $\varepsilon$ is chosen to be very small, see below. Let $C = \sum_{i=1}^{n} \frac{a_i}{2Z} + C_0 = \frac{S}{2Z} + C_0$: Intuitively, $C$ would be the cost of evaluating node $\text{AND}_{n+1}$ when starting with this AND node, and when $\varepsilon$ becomes negligible.
- The bound on the expected evaluation cost is $K = C\left(1 - \frac{S^2}{8Z^2}\right) + \frac{1}{9Z^2}$

To finalize the description of $\mathcal{I}_2$, we define the constants as follows:

- $Z = 10\left((n+1)3^n + n^3\right)M^3$
- $C_0 = \frac{Z}{2Z-S} - \frac{S}{2Z}$, so that $C = \frac{Z}{2Z-S}$
- $\beta = \frac{1-C}{2C}$
- $\varepsilon = \frac{1}{90(n+1)^2Z^2}$

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$: the greatest value in $\mathcal{I}_2$ is $Z$ and $\log(Z)$ is linear in $(n + \log M)$. Because $Z$ is very large in front of $S \leq nM$, we do have that $C$, $C_0$ and $\beta$ are all close to $\frac{1}{2}$. We only use that these constants are all non-negative, and that $\beta \leq 1$ and $C \leq 1$, in the following derivation, where we bound the expected cost of an arbitrary evaluation of the DNF tree. Then, using this derivation, we will prove that $\mathcal{I}_1$ has a solution $I$ if and only if $\mathcal{I}_2$ does.

Let us start with the cost of an arbitrary evaluation of the DNF tree. In such an evaluation, we evaluate some (possibly none) of the first $n$ AND nodes before starting to evaluate node $\text{AND}_{n+1}$. Owing to the dominance property stated in Theorem 2, we can assume that the schedule terminates the evaluation of the node. Then, because $\varepsilon$ is very small, we can compute an approximation of the cost as follows: we assume that the schedule terminates after node $\text{AND}_{n+1}$, because all its leaves have success probability close to 1. We will bound the difference between this approximation and the actual cost later on.

Let $I = \{\text{AND}_{\sigma(1)}, \text{AND}_{\sigma(2)}, \ldots, \text{AND}_{\sigma(k)}\}$ , be the subset, of cardinal $k$, of AND nodes that are evaluated, in that order, before node $\text{AND}_{n+1}$. Let $\overline{Cost}$ be the approximated cost of the schedule (terminating after completion of node $\text{AND}_{n+1}$). To simplify notations, we let $x_i = a_{\sigma(i)}$ for $1 \leq i \leq k$, and let $q_i = 1 - p_{i,1}$ for $i \leq n$. By definition,

$$\overline{Cost} = \sum_{i=1}^{k} \frac{x_i}{Z} \prod_{1 \leq j < i} q_j + \left(C - \sum_{i=1}^{k} \frac{x_i}{2Z}\right) \prod_{1 \leq j \leq k} q_j$$

Note that the cost of node $\text{AND}_{n+1}$ has been reduced from its original value, due to the sharing of the streams whose index is in $I$. To evaluate $\overline{Cost}$, we start by approximating

$$\prod_{1 \leq j < i} q_j = \prod_{1 \leq j < i} \left(1 - \frac{x_j}{Z} - \beta \frac{x_j^2}{Z^2}\right)$$

Let

$$F_i = 1 - \sum_{j=1}^{i-1} \frac{x_j}{Z} - \beta \sum_{j=1}^{i-1} \frac{x_j^2}{Z^2} + \sum_{1 \leq j_1 < j_2 < i} \frac{x_{j_1} x_{j_2}}{Z^2}$$

We have

$$\left| \left( \prod_{1 \leq j < i} q_j \right) - F_i \right| \leq \frac{3^n M^3}{Z^3} \tag{19}$$

To see this, we have kept in $F_i$ all terms of the product $\prod_{1 \leq j < i} q_j$ whose denominators include a factor strictly inferior to $Z^3$. The other terms of the product are bounded (in absolute value) by $M^3/Z^3$, because $\beta \leq 1$ and $M \leq Z$. There are at most $3^{i-1} \leq 3^n$ such terms. Hence the desired bound in Equation (19). Letting

$$G = \sum_{i=1}^{k} \frac{x_i}{Z} - \sum_{1 \leq j_1 < j_2 \leq k} \frac{x_{j_1} x_{j_2}}{Z^2},$$

we prove similarly that

$$\left| \left( \sum_{i=1}^{k} \frac{x_i}{Z} \prod_{1 \leq j < i} q_j \right) - G \right| \leq \frac{n 3^n M^3}{Z^3} \tag{20}$$

Indeed, there are $k \leq n$ terms in the sum, each of them being bounded as before. We deduce from Equations (19) and (20), using $C \leq 1$, that

$$\left| \overline{Cost} - \left( G + (C - \sum_{i=1}^{k} \frac{x_i}{2Z}) F_{k+1} \right) \right| \leq \frac{(n+1) 3^n M^3}{Z^3} \tag{21}$$

Now, we aim at simplifying $H = G + (C - \sum_{i=1}^{k} \frac{x_i}{2Z}) F_{k+1}$ by dropping terms whose denominator is $Z^3$. We have

$$H = \sum_{i=1}^{k} \frac{x_i}{Z} - \sum_{1 \leq j_1 < j_2 \leq k} \frac{x_{j_1} x_{j_2}}{Z^2} + (C - \sum_{i=1}^{k} \frac{x_i}{2Z})(1 - \sum_{j=1}^{k} \frac{x_j}{Z} - \beta \sum_{j=1}^{k} \frac{x_j^2}{Z^2} + \sum_{1 \leq j_1 < j_2 \leq k} \frac{x_{j_1} x_{j_2}}{Z^2})$$

Defining

$$\tilde{H} = C + \frac{1 - 2C}{2Z} \sum_{i=1}^{k} x_i + \frac{1}{2Z^2} (\sum_{i=1}^{k} x_i)^2 + \frac{C - 1}{2Z^2} \sum_{1 \leq j_1 < j_2 < k} x_{j_1} x_{j_2} - \frac{\beta C}{Z^2} \sum_{i=1}^{k} x_i^2,$$

we derive (using $C \leq 1$ and $\beta \leq 1$) that:

$$\left| H - \tilde{H} \right| = \left| \frac{1}{2Z^3} (\sum_{i=1}^{k} x_i) ( \sum_{1 \leq j_1 < j_2 < k} x_{j_1} x_{j_2} + \sum_{i=1}^{k} x_i^2 ) \right|$$

hence

$$\left| H - \tilde{H} \right| \leq \frac{n^3 M^3}{Z^3} \tag{22}$$

Developing $(\sum_{i=1}^{k} x_i)^2 = \sum_{i=1}^{k} x_i^2 + 2 \sum_{1 \leq j_1 < j_2 < k} x_{j_1} x_{j_2}$ in $\tilde{H}$, we obtain

$$\tilde{H} = C + \frac{1 - 2C}{2Z} \sum_{i=1}^{k} x_i + \frac{C}{Z^2} \sum_{1 \leq j_1 < j_2 < k} x_{j_1} x_{j_2} + \frac{1 - 2\beta C}{2Z^2} \sum_{i=1}^{k} x_i^2$$

We have chosen the constants $C$ and $\beta$ so that $\tilde{H}$ can be reduced to

$$\tilde{H} = C + \frac{C}{2Z^2}\left((\frac{S}{2} - \sum_{i=1}^{k} x_i)^2 - \frac{S^2}{4}\right) \tag{23}$$

Indeed, we have $\frac{1-2\beta C}{2Z} = \frac{-SC}{2Z^2}$, and $C = 1 - 2C\beta$. Altogether, we derive from Equations (21) to (23) that

$$\left|\overline{Cost} - C\left(1 - \frac{S^2}{8Z^2}\right) - \frac{C}{2Z^2}\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2\right| \leq \frac{\left((n+1)3^n + n^3\right)M^3}{Z^3} = \frac{1}{10Z^2} \tag{24}$$

Finally, we coarsely bound the difference between the actual cost *Cost* of the schedule and the approximated cost $\overline{Cost}$. The actual probability of evaluating the $i$-th leaf of node $\text{AND}_{n+1}$ is $(1-\varepsilon)^i$ so that the error term for that leaf does not exceed $\left(1-(1-\varepsilon)^i\right)\max(\frac{M}{2Z}, C) \leq n\varepsilon$. Since there are $n+1$ terms, we get a difference bounded by $n(n+1)\varepsilon$. Next we have neglected the evaluation of the remaining AND nodes after node $\text{AND}_{n+1}$, but this cost is (similarly) bounded by $(n+1)\varepsilon\frac{S}{Z} \leq (n+1)\varepsilon$. Altogether, we obtain that

$$\left|Cost - \overline{Cost}\right| \leq (n+1)^2\varepsilon = \frac{1}{90Z^2} \tag{25}$$

Combining Equations (24) and (25), we finally derive that

$$\left|Cost - C\left(1 - \frac{S^2}{8Z^2}\right) - \frac{C}{2Z^2}\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2\right| \leq \frac{1}{9Z^2} \tag{26}$$

We now prove that $\mathcal{I}_1$ has a solution $I$ if and only if $\mathcal{I}_2$ does. Suppose first that $\mathcal{I}_1$ has a solution $I$: $\sum_{i\in I} a_i = \frac{S}{2}$. We evaluate the AND nodes whose indices are in $I$ before evaluating node $\text{AND}_{n+1}$. Let *Cost* be the cost of this evaluation. From Equation (26), we have

$$\left|Cost - C\left(1 - \frac{S^2}{8Z^2}\right)\right| \leq \frac{1}{9Z^2}$$

hence $Cost \leq C\left(1 - \frac{S^2}{8Z^2}\right) + \frac{1}{9Z^2} = K$ thereby providing a solution to $\mathcal{I}_2$.

Suppose now that $\mathcal{I}_2$ has a solution whose cost is $Cost \leq K$, and let $I$ denote the (index) set of AND nodes that are evaluated before node $\text{AND}_{n+1}$. If (by contradiction) we have $\sum_{i\in I} a_i \neq \frac{S}{2}$, then $\left(\frac{S}{2} - \sum_{i=1}^{k} x_i\right)^2 \geq 1$, and Equation (26) shows that

$$Cost \geq C\left(1 - \frac{S^2}{8Z^2}\right) + \frac{C}{2Z^2} - \frac{1}{9Z^2} = K + \frac{9C-4}{9Z^2}$$

Since $9C - 4 = \frac{Z+4S}{2Z-S} > 0$, and $Cost > K$, we obtain a contradiction. Therefore $\sum_{i\in I} a_i = \frac{S}{2}$, and $\mathcal{I}_1$ has a solution. This concludes the proof.

It is interesting to point out that instance $\mathcal{I}_2$ is constructed so that the ordering of the leaves inside each AND node has no importance. In fact, only the last AND node has more than one leaf, and because its leaves have all very high success probability, their ordering does not matter. This shows that the combinatorial difficulty of the DNF-Decision problem already lies in deciding the ordering of the AND nodes. □

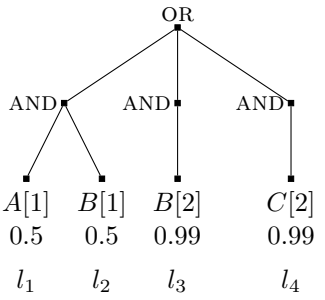



Figure 8: Example DNF tree for which the best schedule has larger cost than a non-linear strategy.

# F Dominant Linear Strategy Counter-Example

A more general notion than a schedule, called a *strategy*, is described in [3]. Although it may seem counter-intuitive, the processing of a query does not have to follow a defined ordering of the leaves. Instead, a strategy is a decision tree in which the next leaf to be evaluated is chosen based on the truth value of the leaves that have been evaluated previously. A *schedule*, as defined in this work, is a particular kind of strategy, termed a "linear strategy" in [3]. Therein, the authors prove that for some problem instances the best linear strategy can be far from being the optimal strategy. Although interesting from a theoretical standpoint, a practical drawback of a non-linear strategy is that the size of its description is exponential in the number of tree leaves. Instead, a linear strategy, or schedule, is simply an ordering of the leaves, with a description size linear in the number of tree leaves. This severe drawback explains why we have not considered non-linear strategies in this work.

However, from a theoretical point of view, it is interesting to ask the following question: while linear strategies are dominant (among all possible strategies) for DNF trees in the *read-once* case [3], is it still the case in the *shared* case? We show that the answer is negative by building a counter-example.

Consider three streams, $A$, $B$, and $C$, with per data item costs $c(A) = 1$, $c(B) = 1.1$, and $c(C) = 1$. Consider the query tree in Figure 8, where for each leaf is indicated the success probability, the stream needed, and the number of data items required from that stream. We first compute the best schedule (i.e., leaf ordering). The cost of schedule $l_1, l_2, l_3, l_4$ is

$$\begin{aligned} c(A) + p_1(c(B) + (1-p_2)c(B)) + (1-p_1)(2c(B)) \\ +(1-p_1p_2)(1-p_3)(2c(C)) = 1.95 < 2 \end{aligned}$$

The cost of any schedule starting with $l_3$ or $l_4$ is at least 2. The cost of schedule $l_1, l_2, l_4, l_3$ is larger than

$$c(A) + p_1(c(B) + (1-p_1p_2)(2c(C)) = 2.15 > 2.$$

Finally, if we start with the first AND node, it is always better to start with leaf $l_1$ whose cost is 0. Altogether, the best schedule is $l_1, l_2, l_3, l_4$, of cost 1.95.

Now, consider the non-linear strategy that evaluates $l_1$ first and then:

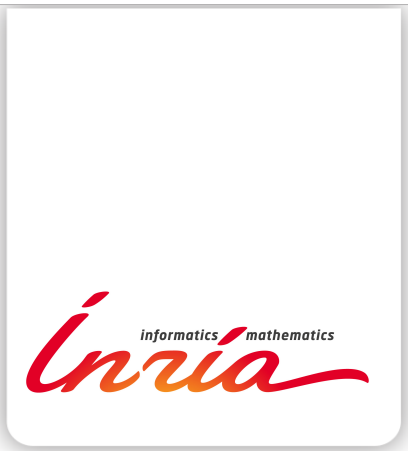
- if $l_1$ evaluates to TRUE, proceeds with $l_2$, $l_3$, and $l_4$, just as in the optimal schedule;
- if $l_1$ evaluates to FALSE, proceeds with $l_4$, $l_3$, and $l_2$.

The cost of this strategy is

$$\begin{aligned} & c(A)+ \\ & p_1[(c(B)+(1-p_2)c(B))+(1-p_2)(1-p_3)(2c(C))] \\ & +(1-p_1)[2c(C))+(1-p_4)(2c(B))] = 1.851, \end{aligned}$$

which is lower than that of the best schedule.

Determining the optimal non-linear strategy for a DNF tree in the *shared* model is an open problem. Unless some structural property of this strategy can be proven, the space required to describe this optimal non-linear strategy is unknown (and likely exponential).




**RESEARCH CENTRE**
**GRENOBLE – RHÔNE-ALPES**

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399